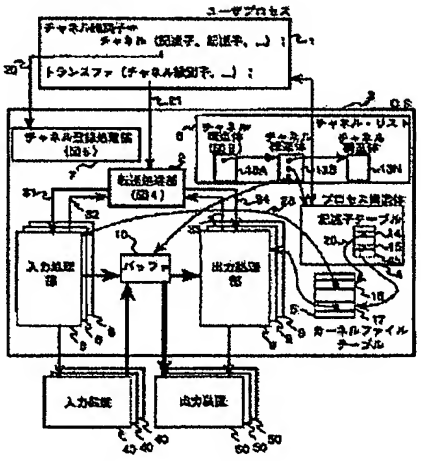
DATA TRANSFER METHOD BY OPERATING SYSTEM

Publication number:	JP8297585
Publication date:	1996-11-12
Inventor:	MASUOKA YOSHIMASA; KAGIMASA TOYOHIKO
Applicant:	HITACHI LTD
Classification:	
- International:	*G06F9/46;* G06F9/46; (IPC1-7): G06F9/46
- European:	
Application number:	JP19950101152 19950425
Priority number(s):	JP19950101152 19950425

Abstract of JP8297585

PURPOSE: To reduce the overhead generated when an OS transfers data from an input source resource to an output destination resource at a request from an application program. CONSTITUTION: A user process 1 specifies a pair of the input source resource and output destination resource used for the data transfer as a channel to the OS, and a channel registering process part 7 registers it in one of channel structures 13A-13N. Further, the user process 1 specifies the channel to be used and the amount of data to be transferred and requests the OS to transfer the data. Under the control of a transfer processing part 2, a buffer 10 is secured in an OS kernel for the channel, an input processing part 8 for accessing the input destination resource and an output processing part 9 for accessing the output destination resource are selected, and the buffer 10 is used repeatedly to repeat the data transfer by the selected input processing part and output processing part until the data are transferred by the specified amount.



(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（A）　(11)特許出願公開番号

特開平8－297585

(43)公開日　平成８年(1996)11月12日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/46	３４０		Ｇ０６Ｆ　9/46	３４０Ａ	

審査請求　未請求　請求項の数11　ＯＬ　（全 26 頁）

(21)出願番号　特願平7－101152

(22)出願日　平成７年(1995)４月25日

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地

(72)発明者　増岡　義政
東京都国分寺市東恋ケ窪１丁目280番地
株式会社日立製作所中央研究所内

(72)発明者　鍵政　豊彦
東京都国分寺市東恋ケ窪１丁目280番地
株式会社日立製作所中央研究所内

(74)代理人　弁理士　薄田　利幸

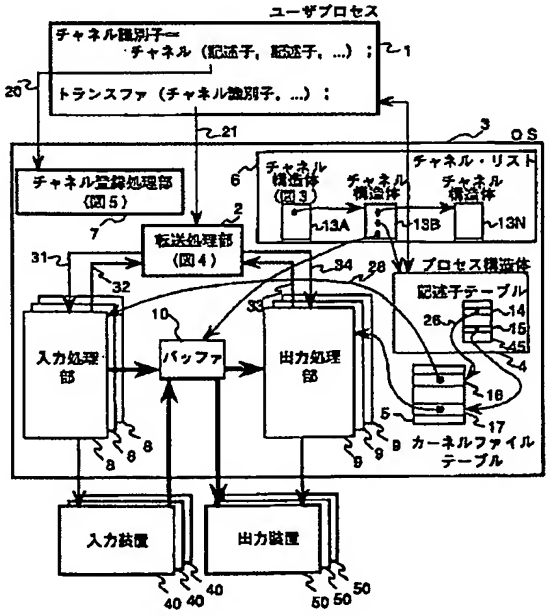
(54)【発明の名称】　オペレーティングシステムによるデータ転送方法

(57)【要約】

【目的】アプリケーションプログラムからの要求にしたがって入力元リソースから出力先リソースへＯＳがデータを転送するときのオーバヘッドを軽減する。

【構成】ユーザプロセス１により、データ転送に使用する入力元リソースと出力先のリソースの対をチャネルとしてＯＳに指定し、チャネル登録処理部７によりこれをチャネル構造体１３Ａ～１３Ｎの一つに登録する。さらに、そのユーザプロセス１より、使用するチャネルと転送すべきデータ量を指定して、ＯＳにデータ転送を要求する。転送処理部２の制御の元で、このチャネル用に、ＯＳカーネル空間内にバッファ１０を確保し、入力先リソースをアクセスするための入力処理部８と出力先リソースをアクセスするための出力処理部９を選択し、選択された入力処理部と出力処理部により、指定された量のデータの転送が終了するまでをこのバッファ１０を繰返し使用してデータ転送を繰り返す。

図１



1

【特許請求の範囲】

【請求項１】処理装置と、主記憶と、複数の入出力装置とを有し、複数のアプリケーションプログラムがオペレーティングシステム（ＯＳ）の制御の下で実行されている計算機システムにおける、オペレーティングシステムによるデータ転送方法であって、

転送すべきデータを保持する入力元リソースと、そのデータを受け取るべき出力先リソースとの対と、該入力元リソースと該出力先リソースとの間で転送すべきデータの量とを指定する、該複数のアプリケーションプログラムの一つから発行されたデータ転送要求に応答して、該入力元リソースに保持されたデータの内、所定量の大きさのデータ部分を該入力元リソースから該主記憶に読み込み、

該読み込みの完了後、該読み込まれたデータを該主記憶から該出力先リソースに書き出し、

該書き出しの完了後、該読み込みと該書き出しを、該データ転送要求で指定された量のデータが該入力元リソースから該出力先リソースに転送されるまで、該入力元リソースに保持された他のデータ部分に対して繰り返し、

該繰返しの終了後に、該アプリケーションプログラムに、該データ転送要求で要求されたデータ転送の終了を通知するステップを有するもの。

【請求項２】該入力元リソースと該出力先リソースの対に対応して、該要求されたデータ転送を制御する情報ブロックを記憶し、

上記入力元リソースからのデータ部分の読み込みあるいは該出力先リソースへの書き出しの実行を該情報ブロックに基づいて制御し、

上記入力元リソースからのデータ部分の読み込みあるいは該出力先リソースへの書き出しを実行する毎に該情報ブロックを書き換えるステップをさらに有する請求項１記載のオペレーティングシステムによるデータ転送方法。

【請求項３】該情報ブロックは、少なくとも、上記データ転送要求で指定された転送データ量、該入力元リソースから読み込み済みのデータの総量、該出力先リソースへ書き出し済みのデータの総量を含み、

上記方法は、

該入力元リソースからの読み込みステップを実行する毎に、該読み込み済みのデータの総量を更新し、

該出力先リソースへのデータの書き出しを実行する毎に、該書き出し済みのデータの総量を更新するステップをさらに有し、

該繰返すステップは、該更新された、読み込み済みのデータの総量が該要求されたデータ転送量に等しくなるまで、該入力元リソースからのデータの読み込みを繰返し、該更新された、書き出し済みのデータの総量が該要求されたデータ転送量に等しくなるまで、該出力先リソースへのデータの書き出しを繰返すステップを有する請

2

求項１記載のオペレーティングシステムによるデータ転送方法。

【請求項４】該読み込みステップは、

該データ転送要求に応答して、該入力元のリソースがデータ読み出し可能な状態にあるか否かを判別し、

該入力元のリソースがデータ読み出し可能な状態にあるときには、上記ＯＳ内に準備された複数の入力処理ルーチンの内、該入力元リソースに対してデータの読み出し動作を実行可能な一つの入力処理ルーチンに対して、該入力元リソースから該主記憶へのデータの読み込みを要求し、

該一つの入力処理ルーチンの制御下で該入力元リソースに対して該要求された読み込みを該実行するステップを有し、

該書き出しステップは、

該出力元リソースがデータを書き込み可能な状態にあるか否かを判別し、

該出力元のリソースがデータ書き込み可能な状態にあるときには、上記ＯＳ内に準備された複数の出力処理ルーチンの内、上記出力元リソースに対してデータの書き込み動作を実行可能な一つの出力処理ルーチンに対して、該読み込みステップで読み込まれたデータの該主記憶から該出力先リソースへの書き出しを要求し、

該一つの出力処理ルーチンの制御下で、該出力先リソースに対して該要求された書き出しを実行するステップを有する請求項１記載のオペレーティングシステムによるデータ転送方法。

【請求項５】該データ転送要求を該一つのアプリケーションプログラムから該ＯＳに発行する前に、その一つのアプリケーションプログラムから発行された、該入力元リソースと該出力先リソースとの対の登録要求に応答して、上記ＯＳ内に準備された複数の入力処理ルーチンの内、該入力元リソースに対してデータの読み込み動作を実行可能な一つの入力処理ルーチンと、上記ＯＳに含まれた複数の出力処理ルーチンの内、上記出力元リソースに対してデータの書き込み動作を実行可能な一つの出力処理ルーチンとを選択し、

選択された一つの入力処理ルーチンを起動するための第１の情報と該入力元リソースを指定する第２の情報と、選択された一つの出力処理ルーチンを起動するための第３の情報と、該出力先リソースを指定する第４の情報を該主記憶に記憶するステップをさらに有し、

該読み出し要求の発行は、該記憶された第１、第２のの情報に基づいて、該一つの入力処理ルーチンに、該入力元リソースからのデータの読み出し込みを要求するステップを有し、

該書き出しは、該記憶された第３、第４の情報に基づいて、該一つの出力処理ルーチンに、該出力元リソースへのデータの書き出しを要求するステップを有する請求項１記載のオペレーティングシステムによるデータ転送方

法。

【請求項6】処理装置と、主記憶と、複数の入出力装置とを有し、複数のアプリケーションプログラムがオペレーティングシステム（OS）の制御の下で実行されている計算機システムにおける、オペレーティングシステムによるデータ転送方法であって、

（a）複数のアプリケーションプログラムが要求した複数のデータ転送を制御する情報ブロックであって、それぞれ、いずれか一つのアプリケーションプログラムを指定する情報と、転送されるべきデータを保持する、該一つのアプリケーションプログラムにより指定された入力元リソースを指定する情報と、そのデータを受けとるべき、該一つのアプリケーションプログラムにより指定された出力先リソースを指定する情報と、その入力元リソースとその出力先リソースとの対に関する、該一つのアプリケーションにより指定された転送データ量とを含むものを記憶し、

（b）該複数の情報ブロックの所定の順に順次選択し、

（c）該ステップ（b）により現在選択された情報ブロックに基づいてデータの読み込みを要求し、その要求ではその情報ブロックに含まれた、入力元リソースを指定する情報が指定する入力元リソースから主記憶へのデータの読み出しを要求し、

（d）該要求されたデータの読み出しが終了した後、その読み出しが完了したデータを、該主記憶から、その情報ブロックに含まれた、出力先リソースを指定する情報により指定される出力先リソースに書き出すことを要求し、

（e）該要求されたデータの書き出しが終了した後、該読み出しが完了したデータの後続のデータの読み出しを要求し、その後続のデータの読み出しが完了する毎に、その読み出された後続のデータの書き込みを要求するように、該読み出しの要求ステップ（c）と該書き込みの要求ステップ（d）とを、該一つの情報ブロックに含まれた転送データ量のデータが該入力元リソースから該出力先リソースに転送されるまで繰返すステップを有し、

ここでステップ（b）は、そのステップ（b）によりすでに選択された一つ又は複数の情報ブロックの各々に基づいて、上記ステップ（c）から（e）が実行するのと並行して、該複数の情報ブロックの内、未選択の複数の情報ブロックを順次選択するように実行されるもの。

【請求項7】該ステップ（a）で選択されたいずれか一つの情報ブロックに基づいて、該ステップ（b）を実行する前に、該選択された情報ブロックが指定する入力元リソースがデータの読み込みが可能な状態にあるか否かを判別し、

その入力元リソースがデータを読み込み可能であるときに上記ステップ（b）をその情報ブロックに基づいて実行し、

その入力元リソースがデータを読み込み可能でないときに上記ステップ（b）をその情報ブロックに対して実行するのを延期し、

該一つの情報ブロックに基づいて、該ステップ（b）を実行した後、該ステップ（c）を実行する前に、該選択された情報ブロックが指定する出力先リソースがデータの書き出しが可能な状態にあるか否かを判別し、

その出力先リソースがデータの書き出しが可能であるときに上記ステップ（c）をその情報ブロックに基づいて実行し、

その出力先リソースがデータを書き出し読み込み可能でないときに上記ステップ（c）をその情報ブロックに対して実行するのを延期するステップをさらに有する請求項6記載のオペレーティングシステムによるデータ転送方法。

【請求項8】各情報ブロックは、それが指定する入力元リソースからデータを主記憶に読み込みが完了したか否かを示す読み込み完了情報と、該入力元リソースから読み出されたデータをその情報ブロックが指定する出力先リソースに書き出し済みか否かを示す書き出し完了情報をさらに有し、

該方法は、

（f）各情報ブロックに基づく、該ステップ（c）またはステップ（e）によるステップ（c）の繰返しにおける、データの読み込みが完了する毎に、その情報ブロック内の、入力元リソースからの読み込みが完了したか否かを示す上記情報をセットし、

（g）その情報ブロックに基づく、該ステップ（d）またはステップ（e）によるそのステップ（d）の繰返しにおけるデータの書き出しが完了する毎に、その情報ブロック内の、出力先リソースへの書き出しが完了したか否かを示す上記情報をセットし、

（h）該複数の情報ブロックが繰返し選択されるように、上記ステップ（a）を繰返し、

（i）各情報ブロックが該ステップ（b）または（h）により選択される毎に、その情報ブロック内の、上記読み込み完了情報および上記書き出し完了情報に基づいて、上記ステップ（c）または（d）を行うべきかを判別するステップをさらに有する請求項6記載のオペレーティングシステムによるデータ転送方法。

【請求項9】該ステップ（h）は、いずれかの情報ブロックに基づいて実行された該ステップ（c）により要求された読み出しの完了ごとにおよび該いずれの情報ブロックに基づいて実行された該ステップ（d）により要求された書き出しの完了ごとに、起動される請求項8記載のオペレーティングシステムによるデータ転送方法。

【請求項10】該ステップ（a）は、

（a1）いずれかのアプリケーションプログラムから発行された、一つの入力元リソースと出力先リソースの対の登録要求に応答して、その対を表わす情報を含む一つのの情報ブロックを記憶し、

5

（ａ２）該一つのアプリケーションプログラムからその後発行された、上記対と転送データ量とを指定するデータ転送要求に応答して、その対に対してステップ（ａ１）で記憶された上記一つの情報ブロック内に、その指定された転送データ量を記憶するステップを有する請求項８記載のオペレーティングシステムによるデータ転送方法。

【請求項１１】各情報ブロックは、データ転送を要求したいずれかのアプリケーションプログラムが指定した該入力元リソースから読み込み済みのデータの総量と、該アプリケーションプログラムが指定した該出力先リソースへ書き出し済みのデータの総量を含み、

上記方法は、

該入力元リソースからの読み込みステップ（ｃ）を実行する毎に、該読み込み済みのデータの総量を更新し、

該出力先リソースへのデータの書き出しステップ（ｄ）を実行する毎に、該書き出し済みのデータの総量を更新するステップをさらに有し、

該繰返すステップ（ｅ）は、該更新された、読み込み済みのデータの総量が該要求されたデータ転送量に等しくなるまで、該入力元リソースからのデータの読み込みステップ（ｃ）を繰返し、該更新された、書き出し済みのデータの総量が該要求されたデータ転送量に等しくなるまで、該出力先リソースへのデータの書き出しステップ（ｄ）を繰返すステップからなる請求項６記載のオペレーティングシステムによるデータ転送方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、計算機システムにおいて、アプリケーションプログラムがオペレーティングシステム（ＯＳ）に、特定のリソースが関与するデータ転送を要求し、ＯＳがそのリソースを用いてデータ転送を実行するデータ転送方法に関する。

【０００２】

【従来の技術】図１３に示すように、代表的な計算機システム３００上では、１つまたはそれ以上のプロセッサや主記憶を有する処理システム３０１内で動作するアプリケーションプログラム３０２が、ＯＳ（3）の管理の下に、磁気ディスク装置３０４上のファイル３０７や、ネットワーク３０５に接続された送受信回路内のバッファ、コンソールディスプレイ３０６内のビデオＲＡＭあるいは同じ計算機上で走行する別のアプリケーションプログラム3･0３などに対して、データを入出力しつつ処理を進めている。以下では、転送すべきデータの入力元として使用される、磁気ディスク装置装置上のファイル、ネットワークに接続された送受信回路内のバッファ、コンソールディスプレイ内のビデオＲＡＭ、アプリケーションプログラムまたはその他の、入力元のデータを保持すると論理的に考えられるリソースを、互いに区別しないで呼ぶときには、これらを「入力元リソース」

6

と呼び、そのデータの出力先としてこれらのリソースのいずれかが使用されたとき、それらのリソースを「出力先リソース」と呼ぶことにする。このようなリソースに対するデータの入出力処理の一つは、入力元リソースからデータを読み出し、読み出されたデータに処理を施すことなく、出力先リソースにそのまま移動することである。以下では、このようなデータ移動をデータ転送とも呼ぶ。

【０００３】近年分散ネットワークコンピューティングや超並列プロセッサ計算機の普及、あるいはマルチメディアデータの利用拡大に伴い、データ転送の高速化が望まれている。

【０００４】従来のデータ転送の典型的な方法の一つが、例えば S. J. Leffler他著 "TheDesign and Implementation of the 4.3BSD UNIX Operating System," Addison-Wesley, PP.169-185 (1989) に詳しく説明されている。ここでは、ユーザプロセスがＯＳに対して、リードシステムコールを発行して、データを所望の入力元リソースからユーザプロセス内のバッファに読み出すことを要求し、この読み出しの完了後に、このバッファ内のデータを所望の出力先リソースに転送することを要求するの及びライトシステムコール発行する。以下、所望のデータが全て転送されるまで、これらのシステムコールの発行を繰り返す。

【０００５】例として、図２のように、ユーザプロセス６０において、磁気ディスク装置１１上のファイル３０７内のデータを、ネットワーク１２に出力する場合を考える。なお、本明細書では、ユーザプロセスとは、ＯＳの管理の下に実行されるプログラムの制御の流れの単位であり、文献によっては「プロセス」「タスク」「スレッド」などとも呼ばれ、アプリケーションプログラムの一部又は全部を構成するものである。また、この例では、ファイル３０７からデータを入力し、ネットワーク１２に出力する場合について述べるが、他の入力先リソースあるいは他の出力先リソースに関しても同様である。なお、ユーザプロセスからＯＳへのサービスの要求は、ユーザプロセスがシステムコールをＯＳに発行することにより行われる。ＯＳでは、ディスパッチャがこのシステムコールを受け付け、どのようなサービスが要求されているか否かを判定し、そのサービスを実行するための、ＯＳ内の特定の処理部を呼び出す。しかし、後述する実施例を含めて、以下の説明では、説明の簡単化のために、ディスパッチャには言及しないで、ＯＳがシステムコールに応答してその特定の処理部を呼び出すと記述する。

【０００６】さて、ユーザプロセス６０は、入力元リソースとしてファイル３０７指定するオープンシステムコールなどのシステムコールをＯＳに対して発行し、ファイル３０７に対する識別子としてＯＳで予め定められた記述子をＯＳから得る。出力先リソースとして使用する

ネットワーク１２に関しても同様である。これらの記述子を得たユーザプロセス６０は、次にファイル３０７に対応する記述子と、ユーザメモリ空間内に設けたデータ転送用のバッファ５０のアドレスとその大きさを引数として指定して、データの読み出しを要求するリードシステムコール５１を発行する。

【０００７】ＯＳ３では、このシステムコール５１を受けて、ファイルシステム８を呼び出す。すなわち、ＯＳ３は、このシステムコールを発行したユーザプロセス１の状態を格納している４内の記述子テーブル４５を、リードシステムコール５１の際に引数として指定された記述子により参照し、この記述子に対応するエントリ１４から、カーネルファイルテーブル５内のエントリ１６のアドレス２６を得る。カーネルファイルテーブル５は、いろいろの入力処理部と出力処理部を起動するためのアドレスを保持するもので、今の例では、エントリ１６内に、上記システムコールで指定された入力元リソースからデータ入力を行なう時に呼び出すべき入力処理部、今の例ではファイルシステム８、のアドレス２８が納められている。ＯＳ３はこのアドレス２８を用いてファイルシステム８に対して関数呼び出しを行なう。

【０００８】ファイルシステム８は、磁気ディスク装置１１上の然るべき領域に対する読み込み命令３１３を発行し、要求されたデータをファイルシステム内に読み出し（３１１）、さらにバッファ５０に読み出されたデータを書き込む（５５）。

【０００９】これでファイルシステム８の動作が終了し、ＯＳ３からユーザプロセス６０へ制御が戻る。ユーザプロセス６０がバッファ５０に入力されたデータをさらにネットワーク１２に出力しようとする場合、ユーザプロセス６０は、ネットワーク１２を指定する記述子と、バッファ５０のアドレス及び大きさとを引数として指定する、書き込み（ライト）システムコール５３を発行する。ＯＳ３では、リードシステムコールの場合と同様にして、プロトコル処理部９が、バッファ５０のデータを読み出し（５８）、然るべきプロトコル処理を施した後、ネットワーク１２へ出力する（３１４）。これでバッファ５０の大きさに相当するデータの転送が終了したが、ファイル３０７にまだ読み込むべきデータが残っている場合は、ユーザプロセスによりシステムコール５１、５３の発行という手順が繰り返される。

【００１０】このような、データ転送の高速化を支援する様々な方法が提案されてきた。例えば、O. Krieger, M. Stumm, and R. Unrau, "The Alloc Stream Facility —A Redesign of Application-Level Stream I/O", IEEE Computer, Vol.27, No.3, pp.75-82 (March 1994) において説明されている、Alloc Stream Facility（ＡＳＦ）では、図２のバッファ５０に相当する記憶領域をカーネルメモリ空間に設けることにより、いずれかの入力先リソースからこの記憶領域に読み出したデータをアプリケーションプログラムがアクセス可能にし、もってそのデータをアプリケーションプログラム内のバッファにコピーする操作を不要にしている。また、R. Govindan and D. P. Anderson, "Scheduling and IPC Mechanisms forContinuous Media", Proceedings of 13th ACM Symposium on Operating System Principles, pp.68-80 (1991) において説明されている、Memory-mapped Streams (MMS) では、入力元リソースのデータを仮想記憶空間上の領域にマップし、入力すべきデータをＯＳにより予め実記憶上に用意させ、これをアプリケーションプログラムにより利用させることにより、アプリケーションプログラムによる、システムコールの発行回数を抑えている。

【００１１】

【発明が解決しようとする課題】図２を用いて説明した従来のデータ転送方法は、次の問題を有する。

【００１２】まず、ユーザプロセスとＯＳとの間で頻繁に制御の行き来が生ずるため、これに由来するオーバヘッドが大きい。すなわち、あるユーザプロセスがあるデータを読み出すためにリードシステムコールを発行した後、そのデータの読み出しが完了すると、ＯＳは、完了割り込みを発生し、その割り込みを処理するルーチンにより、そのデータに対するリードシステムコールを発行したユーザプロセスを見つけ、そのユーザプロセスにデータの読み出しの完了を通知する。そのユーザプロセスは、この完了通知を受けて、出力先リソースへの、そのデータの書き込みを要求するライトシステムコールを発行する。このように、データ転送のために繰り返しユーザプロセスとＯＳとの間で制御の行き来が生ずる。

【００１３】とくに、入力元リソースから出力先リソースへのデータの転送の場合、多量のデータの転送に対して、リードシステムコールとライトシステムコールの対を複数回発行する必要があり、この制御の往来の回数が増大する。ユーザプロセスとＯＳとの間で制御の行き来が生ずると、その際にユーザモード（いわゆる非特権保護モード）とカーネルモード（いわゆる特権保護モード）との間の、計算機のモード切り替えが必要となり、このモード切り替えの際に発生するコンテクストスイッチングのための処理などがオーバヘッドとなる。

【００１４】さらに、マルチタスキング環境では、複数のユーザプロセスが要求する複数のデータ転送の実行を、効率的にスケジュールすることが難しい。この環境下では、各ユーザプロセスは、ＯＳによるスケジューリングにより、プロセッサを割当てられた状態でしか処理を進めることができない。すなわち、上記ユーザプロセスがデータ読み出しのために、リードシステムコールを発行した後、上記データ読み出しの完了までの間に、そのユーザプロセスがプロセッサ待ちの状態になると、上記読み出し完了の割り込みが発生しても、このユーザプロセスは、そのユーザプロセスに実際にプロセッサが再

9

度割り当てるまで、そのデータに対してライトシステムコールを発行することが出来ない。したがって、そのデータの読み出しが終了しているにも拘らず、そのデータに対するライトシステムコールの発行が遅延する。このように、データ転送を要求するシステムコールの発行タイミングが、ユーザプロセスの状態に依存するため、このデータの転送の実行をスケジュールすることは難しい。さらに、この従来技術では、リソースを最大限有効に利用するように、これらのデータ転送をスケジュールすることも難しい。すなわち、いずれかのユーザプロセスから現に発行されたシステムコールが要求する入力元リソースあるいは出力先リソースが利用可能でないためにそのシステムコールを実行できないが、他の空き状態にある入力元リソースあるいは出力先リソースにを使用するシステムコールがまだ他のユーザプロセスから発行されていないという状態が起こり得る。各ユーザプロセスは、連続するデータを構成する複数のデータ部分を読み出すために、複数のリードシステムコールと複数のライトシステムコール交互に発行する。これらのシステムコールの発行時刻は、各ユーザプロセスに依存するうえに、ＯＳはこれらのシステムコールが現に発行された時点でしか、スケジュールすべきシステムコールを知り得ない。したがって、ＯＳは、複数のユーザプロセスにより要求されるデータ転送を効率よく実行するようにスケジュールできない。

【００１５】本発明の目的は、アプリケーションプログラムから要求されたデータ転送を実行する間に生じる、そのユーザプロセスとＯＳの間の制御の往来を減少するデータ転送方法を提供することである。

【００１６】本発明の他の目的は、複数のアプリケーションプログラムが要求する複数のデータ転送の実行をＯＳによりスケジュールし易いデータ転送方法を提供することである。

【００１７】

【問題点を解決するための手段】上記目的を達成するために、本発明では、アプリケーションプログラムにより、入力元リソースと出力先リソースの対（チャネル）と転送すべきデータの量を指定して、それらのリソースの間のデータ転送をオペレーティングシステムに要求し、この要求に応答して、ＯＳにより、この指定された量のデータの転送を実行する。具体的には、ＯＳが使用可能なバッファを確保し、入力元リソースからデータを読み出し、このバッファに読み出されたデータを書き込み、その後この書き込まれたデータをこのバッファーから出力先リソースに転送する。以上の読み出しから転送までの動作を、上記の指定された量のデータが転送されるまで繰り返す。この結果、アプリケーションプログラムは一度のデータ転送要求をＯＳに発行するだけで、後はＯＳが実際にデータ転送を繰返し実行する。したがって、本発明では、ユーザプロセスとＯＳとの間の制御の往来が非常に少なくなる。その結果、この際に必要となるユーザモードとカーネルモードとの間の計算機モードの切り替えに伴うオーバヘッドが減少する。

10

【００１８】さらに、本発明では、複数のアプリケーションプログラムからのデータ転送要求の各々を上記の方法で実行するとともに、それぞれのデータ転送要求のためのデータの読み出し及び書き込みの繰返しをＯＳにより制御する。この結果、複数のアプリケーションプログラムプログラムが、複数の異なるチャネルにおけるデータ転送を要求しているような環境においても、これらのデータ転送の実行をＯＳによりスケジュールしやすくなる。

【００１９】

【実施例】以下、本発明に係るデータ転送方法を図面に示した実施例および変形例を参照してさらに詳細に説明する。

【００２０】＜実施例１＞図１において、４０は、入力元リソースとして使用され得る入力装置を表わし、例えば、磁気ディスク装置、他の計算機システム等に接続されたネットワーク、キーボードなどを示す。５０は、出力先リソースとして使用され得る出力装置を表わし、例えば、上に述べた磁気ディスク装置と同じかもしくは異なるディスク記憶装置、上のネットワークと同じネットワーク、あるいは他のキーボードなどを示す。

【００２１】ユーザプロセス１は、データ転送をＯＳ３に要求するときに、従来と同様に、使用する入力元リソースと出力先リソースの記述子をＯＳに決定して貰い、データ転送の入力元リソースの記述子と出力先リソースの記述子との対を指定するようになっている。以下では、データ転送すべき入力元リソースと出力先リソースの組合せを「データ転送路」もしくは「チャネル」と呼ぶ。２０は、ユーザプロセスが使用する新たなシステムコールとして用意されたもので、入力元リソースに対応する記述子と出力先リソースに対応する記述子との対を引数として含み、この対により指定されたチャネルを登録することをＯＳ３に要求するチャネルシステムコールである。ＯＳ３では、チャネル登録処理部７が、このシステムコールに応答して、このチャネルに関する情報を保持するチャネル構造体を作成することにより、そのチャネルを登録する。さらに、そのチャネルに対してチャネル識別子を決定し、ユーザプロセス１にこのシステムコールの戻り値としてこのチャネル識別子を返す。チャネル構造体１３Ａ～１３Ｎは、登録されたいろいろのチャネルの現在の状態を保持するもので、それらは互いにチェインされて、チャネルリスト６が形成される。このチャネルシステムコールは、さらに、そのチャネルを介したデータ転送に使用するバッファの大きさを指定するようになっており、チャネル登録処理部７は、この指定された大きさに基づいて、バッファのサイズを決め、さおの決められたサイズのバッファとして、カーネルアド

レス空間（カーネル空間）内に位置し、このＯＳが走行する計算機システムの主記憶（図示せず）内の領域が割り当てられたバッファ１０を確保する。さらに、その決められたバッファサイズをそのチャネル構造体内に書き込む。

【００２２】２１は、ユーザプロセス用に設けられた他のシステムコールで、これは、データ転送に使用するチャネル識別子と転送データ量とを引数として含み、この識別子で指定されるチャネルにおけるデータ転送をＯＳ３に要求するトランスファシステムコールである。ＯＳ３では、転送処理部２がこのトランスファシステムコールを受け付け、チャネルリスト６内の、このシステムコールを発行したユーザプロセスに対して設けられたチャネル構造体を参照しながら、要求されたデータ転送を行なう。すなわち、本実施例では、ユーザプロセスが入力元リソースと出力先リソースの対を指定するチャネルし、この入力先リソース内の、ユーザプロセスが指定したデータの全てを、この出力先リソースへ転送し終るまで、データ転送を入力元リソースと出力先リソースの間で繰返し実行するように、転送処理部２がデータ転送を制御するところに特徴がある。この結果、ユーザプロセスは、一度データ転送を要求するシステムコールをＯＳを発行すればよい。したがって、従来のようにデータ転送のためにユーザプロセスとＯＳとの間で頻繁に制御の行き来が生じるようなことがないので、データ転送に付随するオーバヘッドが小さくなる。

【００２３】より具体的には、ＯＳ（３）は、さらに、従来技術と同様に、複数の入力元リソースからのデータの読み出しを行なうための複数の入力処理部８と、複数の出力先リソースへのデータの書き込みを行なうための複数の入力処理部９とを有し、転送処理部２は、そのシステムコールが指定するチャネルに属する入力元リソースと出力先リソースを、そのシステムコールを発行したユーザプロセスに対応して設けられたチャネル構造体にもとづいて判別し、その入力元リソースからの一定量のデータの読み出しを、その入力元リソースに体する入力動作を行い得る、いずれか一つの入力処理部８に要求し、この入力処理部８の制御により、そのデータがそのチャネル用に設けたバッファ１０に読み出されると、そのチャネルに属する出力先リソース用に設けられたいずれかの出力処理部９に、その読み出されたデータをその出力先リソースに書き込むことを要求する。この書き込みが終了後、上記読みだされたデータの後続のデータに関して、データの読み出しと書き込みを、上記トランスファコールで指定された転送データ量のデータが転送されるまで繰り返す。

【００２４】本実施例では、複数のユーザプロセスが複数のチャネルをＯＳに登録してデータ転送を行なわせることができる構造となっている。しかし、図１では、簡単化のために、チャネル構造体以外は、１つのユーザプロセスが登録した、１つのチャネルに関連する部分のみを図示し、その他は省略している。

【００２５】転送制御部２は、以上のデータ転送を複数のユーザプロセスから要求された複数のデータ転送の各々に対して実行し、かつ、これらの複数のデータ転送を並列に実行するように、これらのデータ転送のためのデータの読み込みとデータの書き出しをスケジュールする。このために、本実施例では、転送制御部２は、チャネルリスト６を繰返し走査して、それぞれのチャネル構造体が指定する、入力元リソースからのデータ入力あるいは出力先リソースからのデータ出力のいずれを実行すべきかを判定し、その実行すべきデータ入力あるいはデータ出力を実行する、然るべき入力処理部８または出力処理部９に、入力要求あるいは出力要求を発行する。この入力要求あるいは出力要求に基づいて、データの読み込みがそのチャネルに体して実行されているのと並行して、転送制御部２は、チャネルリスト２の後続のチャネル構造体に対して、上に述べたのと同じ処理を実行する。

【００２６】具体的には、転送制御部２は、いずれかの走査されたチャネル構造体が指定するチャネルに関して、その入力元リソースからそのチャネル用のバッファへのデータの読み出しがなされていないときには、データの読み出しを要求し、その入力元リソースからそのバッファへのデータの読み出しが終了している場合には、そのデータの、そのバッファから出力先リソースへの書き出しを要求する。さらに、このデータの書き出しが終了している場合には、再度、入力元リソースからそのバッファへの後続のデータの読み出しを要求する。こような制御のために、各チャネル構造体には、データの読み込みが終了したか否かの情報と、データの書き出しが終了したか否かの情報と、ユーザプロセスに要求された転送データ量、すでに読み込まれたデータの総量、すでに書き出されたデータの総量などを保持するようになっていて、そのチャネル構造体が代表するチャネルに関して、データの読み込みあるいはデータの書き出しが終了した時点で、これらの情報を更新するようになっている。

【００２７】本実施例では、チャネルリストを走査する契機は、いずれかのユーザプロセスから新にトランスファコールが発行されたことがチャネル登録部７から通知された場合、あるいは、いずれかのチャネルに関して、データの読み込みが終了したことがいずれかの入力処理部８から通知された場合、あるいは、データの書き出しが終了したことがいずれかの出力処理部９から通知された場合である。

【００２８】したがって、本実施例では、複数のデータ転送の間に頻繁に発生する呼び出しは、転送処理部２から入力処理部８の呼び出し３１、出力処理部９の呼び出し３４、及びそれらの逆である入力処理部８から転送処

理部２の呼び出し３２および出力処理部９から転送処理部２の呼び出し３３であるが、これらはすべて、システムコールのような割込み（トラップ）ではなく、サブルーチン呼び出しで呼び出しであることに注意すべきである。これにより、これらのデータ転送を小さいオーバヘッドで行なうことができる。

【００２９】以上のごとく、本実施例では、複数のチャネルにおけるデータ転送をＯＳ内で制御することができる。この結果、たとえ、データ転送を要求したユーザプロセスが、その後、プロセッサ待ちの状態になっても、ＯＳは、そのユーザプロセスがすでに要求したデータ転送を継続することが出来る。また、複数のチャネルにおいてデータ転送が行なわれている場合でも、１つ１つのチャネルにおけるデータの流れを容易に制御できる。

【００３０】なお、本実施例でも、図２に示した従来技術と同様に、複数の入力処理部８と複数の出力処理部のアドレスを保持するカーネルファイルテーブル５と、各プロセスに関する情報を保持するプロセス構造体４とが設けられている。この４には、従来と同様に、記述子テーブル４５が設けられている。この記述子テーブル４５には、このプロセスが使用するチャネルに対する入力動作を実行する一つの入力処理部８のアドレスを保持する、カーネルファイルテーブル内のエントリ１６のアドレス１４と、そのチャネルに対する出力動作を実行する一つの出力処理部９のアドレスを保持する、カーネルファイルテーブル内のエントリ１７のアドレス１１５、および、その記述子に対して次に入出力動作を行おうとする時に、入出力を行う位置を示すファイルポインタとを保持する。以下、本実施例の動作を詳細に説明する。

【００３１】（記述子テーブルとカーネルファイルテーブル）ユーザプロセス１は、データ転送をＯＳに要求する前に、以下のような処理を従来技術と同様に実行する。

【００３２】ＯＳは予めカーネルファイルテーブル５を従来技術と同様に、予め作成する。このテーブル５は、各入力処理部８と各出力処理部９を起動するのに使用する、それぞれの処理部のアドレスを保持している。また、各ユーザプロセスは、自分の現在の状態を保持するプロセス構造体をＯＳ内に持っている。

【００３３】ユーザプロセス１が使用する入力元リソースまたは出力先リソースの記述子の決定は以下のように従来と同様にして行われる。

【００３４】走行中のユーザプロセスの各々には、予め、例えば、そのユーザプロセスの起動時にＯＳ３により４が割り当てられる。いずれかのユーザプロセスが、いずれかのディスク記憶装置に保持されたファイル内のデータを利用するとき、そのユーザプロセスは、そのファイルの名称と、ファイル内のデータの内、使用すべきデータの先頭アドレスを指定するオープンシステムコールというシステムコールをＯＳに発行する。ＯＳはこのシステムコールに応答して、このファイルに対する記述子を決定して、ユーザプロセスに通知するとともに、この記述子に対応して記述子テーブル４５の１つのエントリ１４を割当てる。このエントリ１４は、このファイルを保持するディスク装置をアクセスするために用意された、例えば、ファイルシステムと呼ばれる、一つの入出力処理部８のアドレスを保持する、カーネルファイルテーブル５内のエントリ１６のアドレスを保持する。さらに、エントリ１６は、オープンシステムコールで指定された、このファイルの名称を保持している。およびファイル内の読み出すべきデータの先頭アドレスは、記述子テーブル４５内のエントリ１４に、既に述べたファイルポインタとして格納される。このファイルポインタは、そのファイルのどの位置から入力すべきかを示す情報として、入力処理部８（この例ではファイルシステム）により参照される。次に同様にして、上記ユーザプロセスは、例えば上のファイルのデータを他のファイルに転送したい場合、そのファイルに関しても同様なオープンシステムコールをＯＳに発行する。このシステムコールも同様に処理され、記述子テーブル４５には、同様に、この他のファイルに関する、テーブル５のエントリ１７のアドレス１５が格納される。

【００３５】（チャネル構造体）チャネル構造体１３は、入力元リソースから出力先リソースへの１つのデータ転送の、現時点における状態が格納されるデータ構造である。図３に示すように、チャネル構造体１３は以下の情報を格納する記憶領域を含んでいる。

【００３６】フラグ１４１は、そのチャネルの状態を数値化し、要約して示すものである。数値化されるべき状態の主なものは、次の通りである。すなわち、ユーザの要求したデータ転送が終了したことを示すフラグCHDONE、チャネルにおいてデータ転送を行なっている際にエラーが発生したことを示すフラグCHERROR、チャネルシステムコール２０によって登録された後、トランスファシステムコール２１によって転送開始を要求されるのを待っている状態であることを示すフラグCHWAIT、入力元リソースからこれ以上データを読み込めないことを示すフラグCHEOFなどである。ここでフラグCHDONE、CHERROR、CHWAIT、CHEOFなどは、同時にセットされても識別できるような、互いに異なるフラグである。

【００３７】リストポインタ１４２は、別のチャネル構造体へのポインタを格納する。このリストポインタ１４２を用いてすべてのチャネルを結合し、チャネルリスト６を構成する。リストポインタ１４２の値を一定の値（例えば０）にすることにより、このチャネル構造体がリストの最後であることを示すことができる。

【００３８】プロセスポインタ１４３は、このチャネルを登録したユーザプロセスの、プロセス構造体４へのポインタを格納する。

【００３９】入力記述子１４４は、バッファ１０へデー

タの入力を行なう時の入力元リソースを指定する記述子を格納する。この入力記述子１４４と、プロセスポインタ１４３により、このチャネルにおいてデータの入力を行なう際に、呼び出すべき入力処理部８を見出すことができる。

【００４０】出力記述子１４５は、バッファ１０に置かれたデータを出力する時の出力先リソースを指定する記述子を格納する。転送処理部２は、出力記述子１４６と、プロセスポインタ１４３により、このチャネルにおける出力処理部９を見出すことが出来る。

【００４１】チャネル識別子１４６は、ユーザプロセスがトランスファシステムコール２１によりデータ転送を要求する際に、データ転送を実行すべきチャネルを指定するのに用いられる。

【００４２】バイトカウンタ１４７は、データ転送を行なうべきバイト数を格納する。この値が負の数である場合、入力元リソースから入力できるデータがなくなるまでデータ転送を行なうことを示す。

【００４３】バッファ管理構造体１５０は、チャネル構造体１３に含まれるデータ構造で、そのチャネルにおけるデータ転送のために割り当てられたバッファ１０を管理するためのものである。本実施例では、バッファ１０はＯＳ３のカーネル空間内に静的に割り当てられた連続した記憶領域とするが、バッファ１０は、入力元リソースから入力したデータを出力先リソースに出力するまでの間一時的に格納しておくことができれば十分なので、メモリ空間を利用してバッファ１０を実現する方法としては、記憶領域を動的に割り当てるやり方など、本実施例とは異なるバッファ管理の方法を用いることもできる。本実施例では、バッファ管理構造体１５０は、以下のような情報を含む。

【００４４】バッファポインタ１５１は、このチャネルにおけるデータ転送のために割り当てられた記憶領域であるバッファ１０へのポインタを格納する。データポインタ１５２には、バッファ１０上の次に出力すべきデータのアドレスを格納する。入力データカウンタ１５３は、これまでに入力したデータのバイト数を保持する。さらに、出力データカウンタ１５４は、これまでに出力したデータのバイト数を保持する。

【００４５】（チャネルシステムコール）本実施例においては、ユーザプロセスがデータ転送をＯＳに要求するときに、チャネルシステムコールにより、入力元リソースと出力先リソースとの対を指定してチャネルを登録する必要がある。チャネルシステムコールは、次のような形式である。

【００４６】チャネル（入力記述子，出力記述子，バッファサイズ）

ここで、入力記述子は入力元リソースに対応する記述子、出力記述子は出力先リソースに対応する記述子である。これらの記述子は、ユーザプロセス１が利用予定の入力元リソースと出力先リソースをＯＳに指定し、それらのリソースに対する記述子をＯＳから予め得ておく。これらの記述子をＯＳから得るには、従来技術で記載したような公知の方法を使用すればよい。バッファサイズは、このチャネルにおけるデータ転送のために確保すべきバッファ１０の大きさである。

【００４７】（チャネル登録処理部７）ＯＳ３では、このチャネルシステムコールを受けて、チャネル登録処理部７を呼び出す。チャネル登録処理部７は、図５に示すように、新しいチャネル構造体１３をカーネル空間内に確保する（ステップ１２１）。具体的な領域確保の方法は様々であり、ＯＳがサポートしているメモリ割り当てルーチンを用いてもいいし、あらかじめ大きな領域を確保しておき、そこをチャネル構造体のプールとして用いても良い。次にこのチャネル構造体を、発行されたチャネル・システム・コールの引数をもとに初期化する（ステップ１２２）。初期化としてはプロセス・ポインタ１４３、入力記述子１４５及び出力記述子１４６の設定などを含むが、特に、チャネル構造体のフラグ１４１の部分には、フラグCHWAITをセットし、このチャネルはトランスファ・システム・コールによるデータ転送の開始を待っていることを明示する。次に、このチャネル構造体をチャネルリスト６に追加する（ステップ１２３）。リスト６のどこにこのチャネル構造体を追加するかは様々である。次に、データ転送用のバッファ１０のための領域をカーネル空間内に確保し、そのアドレスをチャネル構造体のバッファ・ポインタ１４４に格納する（ステップ１２４）。より具体的な領域確保の方法はチャネル構造体を割り当る時と同様に様々である。ＯＳは、このバッファのサイズを、チャネルシステムコマンドで指定されたサイズを参考にして決める。バッファサイズが指定されない場合、指定されたバッファサイズが不適切な場合、あるいは性能上より好ましいバッファサイズがある場合には、ＯＳは、チャネルシステムコマンドで指定されたサイズとは異なるサイズのバッファをカーネル空間内に確保することもありうる。最後に、このチャネルを識別できる識別番号を１つ決定し、それをこのチャネルの識別子としてチャネル構造体のチャネル識別子１４７に格納する。その後、この値を戻り値としてユーザプロセスに通知する（ステップ１２５）。なお、チャネルシステムコールの引数が不正であるなどの原因によりチャネルの登録に失敗したときは、システムコールの戻り値を－１としてその旨をユーザプロセスに通知する。

【００４８】このチャネル識別子は、その後ユーザプロセスがデータ転送をＯＳ３に要求する時に、転送を行なうべきチャネルを指定するのに用いられる。１つのユーザプロセスは、複数のチャネルを登録できる。チャネル識別子の役割は、トランスファシステムコールを受けた転送処理部２がチャネル（より正確にはチャネル構造体）を一意に識別できるようにするためのもので、この

条件を満たす限りどのようなチャネル識別子の選び方をしてもかまわない。考えられる方法の1つとして、チャネル識別子を、記述子の1つとして扱うというやり方が考えられる。この場合、チャネル識別子を決めるには、従来のOSに備わったルーチンを呼び出して、使用されていない（入出力先リソースが対応付けられていない）記述子を見出し、それをチャネル識別子とすれば良い。こうすることにより、オペレーティングシステムに従来から備わる、記述子を操作するシステムコールが、チャネルに対しても有効となるという利点がある。例として、従来技術で挙げた代表的なオペレーティングシステムが備えるフォークシステムコールにより、新たにユーザプロセスを生成する場合を考える。フォークシステムコールでは、新しいユーザプロセス（子プロセス）は、元のユーザプロセス（親プロセス）の記述子をそのまま受け継ぐことができるという機能を持つ。そのため、親プロセスが登録したチャネルを、チャネルを引き継ぐ機能をOSに新設することなく、新しいユーザプロセスに受け継がせることが可能となる。

【0049】（トランスファシステムコール）ユーザプロセス1がOS3にデータ転送を開始させるためには、トランスファシステムコール21を発行しなくてはならない。このトランスファシステムコール21は次のような形式である。

【0050】トランスファ（チャネル識別子，バイト数）

ここでチャネル識別子はユーザプロセス1が先にチャネルシステムコール20により登録したチャネルの識別子である。また、バイト数は転送すべきバイト数を指定する。なお、本実施例では、データ転送するバイト数を、トランスファシステムコールが発行されてから出力先リソースに出力されたデータのバイト数とする。入力元リソースから入力すべきデータがまだ存在する場合でも、このバイト数だけのデータ転送が行なわれたらトランスファシステムコールは終了する。バイト数として0を指定した場合は、入力すべきデータがなくなるまで（例えば入力元リソースがファイルの時はファイルの終わりまで）データ転送が続けられる。後に説明するように、データ転送処理部2は、トランスファシステムコール21に応答して、データ転送を実行し、実際に転送したバイト数を戻り値として返る。この戻り値が引数で指定したバイト数と同じ場合は、ユーザプロセスの要求したデータ転送が成功裡に終了したことを意味する。また、戻り値が引数よりも小さい場合は、データ転送において何かしら障害があったことを意味し、また負の数である場合は、引数が不正であるなど、データ転送の前にエラーがあったことを示す。

【0051】（データ転送処理部2）トランスファシステムコール21を受けたOS3は、転送処理部2を呼び出す。この転送処理部2のより詳細な構造は、図4のようになっている。

【0052】転送開始／終了処理部73は、トランスファシステムコール73を受け、すでにチャネル登録処理部7によって登録されているチャネルのいずれかにおけるデータ転送を準備し、スケジュール処理部75を呼び出し（84）、そのチャネルでのデータ転送を開始させる。その後、待機処理部74を呼び出し（82）、ユーザプロセスが要求したデータ転送が終了するのを待たせ、データ転送が終了した時点でユーザプロセスに制御を戻す。

【0053】待機処理部74は、基本的にはデータ転送が終了するまでトランスファシステムコールを発行したユーザプロセスを休眠状態にするためのルーチンである（本実施例では行なわないが、ユーザからのオプション設定により、休眠状態にしないようにすることもできる）。しかし、必要に応じて走行可能状態とされて、入力処理部または出力処理部を行なう（86または87）場合もある。

【0054】スケジュール処理部75は、新にトランスファシステムコールがユーザプロセスにより新にいずれかのチャネルに対するデータ転送が要求されたときに転送開始／終了処理部76により起動されるか、あるいは、いずれかの入力処理部8あるいは出力処理部9による入力処理あるいは出力処理が終了したときに、入力終了処理部78あるいは出力終了処理部79により起動される。どれか1つのチャネルに関する事象の発生（具体的には、データ転送要求の発生、データ入出力の終了）により呼び出される。

【0055】スケジュール処理部75の基本的な動作は、チャネルリスト6を走査し、そこに登録されたチャネル構造体の各々を参照して、それぞれが対応するチャネルの現時点での状態を調べ、それぞれチャネルに対しデータの入力とデータの出力の一方あるいは両方が可能かどうかを判定し、データ入力が可能であれば入力要求発行処理部76を呼び出し（88）、データ出力が可能であれば出力要求発行処理部77を呼び出す（90）というものである。

【0056】入力要求発行処理部76及び出力要求発行処理部77は、スケジュール処理部75により指定された1つのチャネルに対して入力要求あるいは出力要求を発行する。より具体的には、そのチャネルにおける入力元リソースまたは出力先リソースへのアクセスを行なう際に呼び出すべきいずれかの入力処理部8あるいはいずれかの出力処理部9を決定し、この決定された入力処理部8あるいは出力処理部9に対して入力要求あるいは出力要求を出力する。これらのルーチンは入力要求または出力要求を発行するだけで、データのバッファ10への入力あるいはバッファ10からのデータの出力が終了するまで待たずにスケジュール処理部75に制御を戻す。この結果、すべてのチャネル構造体の操作を素早く行な

えるため、複数のチャネルを一括してスケジュールするのが容易になる。

【００５７】なお、複数の入力処理部８あるいは複数の出力処理部９の中には、入出力のために呼び出しを受けた場合、データの入出力が終了するまで呼び出したルーチンに制御を返さないという同期入出力しかサポートしていないものがある。

【００５８】本実施例では、こうした入出力処理部に対して入出力要求を発行する場合は、待機処理部７４にて休眠状態となっているユーザプロセスを走行可能状態とし、そこから入力処理部８あるいは出力処理部９を呼び出させることにより、入力要求発行処理部７６及び出力要求発行処理部７７は要求を発行するだけで、データの入出力を待たずにすぐに戻るというセマンティクスを実現する。

【００５９】なお、入力処理部あるいは出力処理部の中には、入出力のために呼び出しを行なった場合、データの入出力が終了するまで呼び出したルーチンに制御を返さないという同期入出力しかサポートしていないものがある。そこで、本実施例では、こうした入出力処理部に対して入出力要求を発行する場合は、待機処理部７４にて休眠状態となっているユーザプロセスを走行可能状態とし、そこから入力処理部あるいは出力処理部を呼び出させることにより、入力要求発行処理部７６及び出力要求発行処理部７７は要求を発行するだけで、データの入出力を待たずにすぐに戻るというセマンティクスを実現する。これらの点を含め、入力要求発行処理部７６及び出力要求発行処理部７７のより詳細な実装については後述する。なお、以下本明細書では、OSが備える各種の入力処理部及び出力処理部を、入出力要求を発行した際にデータの入出力を待たず、すぐに呼び出したルーチンにもどるというセマンティクスをサポートするものと、このようなセマンティクスをサポートせず、入出力要求を発行したら、データの入出力が終了するまで呼び出した入出力処理部内で待つことを余儀なくされるものの２つに分類し、前者を非同期的、後者を同期的であると呼ぶ。

【００６０】入力終了処理部７８及び出力終了処理部７９は、あるチャネルに属する入力元リソースまたは出力先リソースへのアクセスを行なうための入力処理部８または出力処理部９から、バッファ１０へのデータの入力あるいはバッファ１０からのデータの出力が終了した際に呼び出される（９８または９９）。これらのルーチンは、入力元リソースあるいは出力先リソースによって異なる入力処理部８または出力処理部９が、データの入出力が終了した時に返してくる情報を検査し、対応するチャネル構造体が保持している情報を更新し、その後スケジュール処理部７５を呼び出す。これにより、あるチャネルの状態が変化したとき、常にスケジュール処理部７５が呼び出されることになる。

【００６１】待機処理部７４は、基本的にはデータ転送が終了するまでトランスファシステムコールを発行したユーザプロセスを休眠状態にするためのルーチンである。なお、ユーザからのオプション設定により、休眠状態にしないようにすることもできる。しかし、必要に応じて走行可能状態として、入力処理部８または出力処理部９を動作させる（８６または８７）場合もある。

【００６２】以上から明らかなように、転送処理部２において、スケジュール処理部７５は、少なくとも１つのチャネルにおいてその状態が変化したとき常に呼び出され、またあるチャネルにおいてデータの入出力を行おうとする際にも、しかるべき入力処理部８又は出力処理部９を呼び出して入出力要求を発行するだけで、その入出力の実行を待たずに直ちに次のチャネルに対するデータ転送要求の処理へ移行できる。従って、複数のチャネルが登録されてデータ転送が行われている場合でも、実際のデータの流れをスケジュール処理部７５において容易に管理することができる。そのため、スケジュール処理部７５には、必要に応じて、後述するようなアルゴリズム以外にも、リアルタイムスケジューリングアルゴリズムなどの様々なアルゴリズムを容易に実装することができ、それにより本実施例のデータ転送機能をさらに高機能かつ高性能のものとすることができる。

【００６３】（転送開始／終了処理部７３）転送開始／終了処理部７３の動作を図６に示す。この処理部７３は、トランスファシステムコール２１を受けて呼び出され、まずシステムコールの引数として指定されたチャネル識別子に対応するチャネル構造体を、チャネルリスト６を走査して得る（ステップ２２１）。次に、発行されたトランスファシステムコールの引数をチェックしながら、それらをチャネル構造体に設定していく（ステップ２２２）。特に、ここでチャネル構造体のフラグ１４１においてフラグCHWAITをリセットし、データ転送中であることを明示する。また、注意すべき点として、ここで入力データカウンタ１５３及び出力データカウンタ１５４の初期化は行なわない。これらが初期化されるのはチャネル登録処理部７においてのみであり、これにより、データ転送を（シグナルの配送などの原因で）一時中断して再開する場合も、同じトランスファシステムコールを発行することで行なうことができる。以上までで、不当な引数が用いられているなど、何かエラーがあった場合は（ステップ２２３）、それを通知する形で（すなわちトランスファシステムコールの戻り値が－１を返すように）制御がユーザプロセスに戻される（ステップ２２７）。エラーがなかった場合は、スケジュール処理部７５が呼び出される（（ステップ２２４）。スケジュール処理部７５はチャネルリスト６につながれたすべてのチャネル構造体を走査するので、この時点でこのチャネルに対するデータ転送が開始される。スケジュール処理部７５から戻ったら、次に待機処理部７４を呼び出し、ト

ランスファシステムコールにより要求されたバイト数のデータ転送が終了するまで待機する（ステップ２２５）。その後ユーザプロセスに、実際に転送されたデータのバイト数を戻り値として戻る（ステップ２２６）。

【００６４】（待機処理部７４）このルーチンの動作は図７の通りである。このルーチンはある１つのチャネル構造体とともに呼び出される。まずこのチャネル構造体のフラグ１４１においてフラグCHERRORがセットされているかが検査され、このチャネルに対してエラーが発生しているかどうかが調べられる（ステップ２０１）。もしセットされていれば、発生したエラーに関する情報がユーザプロセスに渡るようにして自分を呼んだルーチンに戻る（ステップ２０２）。CHERRORがセットされていなければ、次にフラグCHDONEを検査し、データ転送が終了しているかどうか調べる（ステップ２０３）。CHDONEがセットされている時は、ユーザの要求したデータ転送が完了した旨の通知と共に自分を呼び出したルーチンに戻る（ステップ２０４）。もしCHDONEもCHERRORもセットされていなければ、次に待機処理部はこのチャネルに対して同期入力あるいは同期出力の要求が生じていないかどうか調べる（ステップ２０５および２０７）。これらの検査はチャネル構造体のフラグ１４１において、それぞれフラグCHSYNCRD、CHSYNCWRがセットされているかどうかを調べることにより行なわれる。これらのフラグはチャネル構造体のフラグ１４１において既述のCHDONEなどのフラグとは独立にセットまたはリセットできる。もしフラグCHSYNCRDがセットされている場合は、そのチャネルに対する入力処理部８を呼び出して、バッファ１０に入力元リソースからデータを入力する（ステップ２０６）。そのチャネルに対する入力処理部８を決定するのは、従来の方法であった図２の時と同様、４の記述子テーブル１８、カーネルファイルテーブル５を参照することにより行なわれる。記述子などの情報は、チャネル構造体に格納されている。ステップ２０６の終了時点でデータの入力は（エラーであれなんであれ）終了しているので、次に入力終了処理部７８を呼び出す（ステップ２０８）。その後、待機処理部７４の始めに戻る。CHSYNCWRがセットされている場合も同様に、出力処理部９を呼び出してデータの出力を行ない（ステップ２０９）、続いて出力終了処理部を呼び出し（ステップ２１０）、待機処理部７４の始めに戻る。CHSYNCRD、CHSYNCWRのいずれもセットされていない時は、ＯＳのプロセス管理機構を呼び出して、自分自身を休眠状態にする（ステップ２１１）。

【００６５】ステップ２１１にて休眠状態となったユーザプロセスは、必要に応じて実行可能状態とされる。実行可能状態となったユーザプロセスは、その時点から処理を再開し、ステップ２０１に戻る。ここで注意すべき点は、このユーザプロセスは既にＯＳ内の待機処理部７４にて休眠しており、処理再開後、必要に応じて例えば入力処理部などを直接呼び出すという点である。この間、ＯＳとユーザプロセスとの間の制御の往来、すなわちカーネルモードとユーザモードとの間の計算機モードの切り替えを要しないため、一々システムコールを発行して入出力を行う場合と比較して、オーバヘッドが非常に小さい。

【００６６】（スケジュール処理部７５）この動作を図８および図９に示す。ここではチャネルリスト６上のすべてのチャネル構造体が検査され、データ転送が終了したか、あるいは入出力を行なうことが可能か、などが１つ１つのチャネルに対して調べられる。スケジュール処理部７５においては、まず一連の操作において入力要求または出力要求が発行されたチャネルがあるかどうかを示す変数（Ｉとする）に０を代入して初期化する（ステップ１０１）。次に、チャネルリスト６の先頭にあるチャネル構造体を取り出す（ステップ１０２）。そして、このチャネル構造体を参照して入出力が可能かどうかの検査を行なっていくが、ここでまず始めに、このチャネルにおけるデータ転送を要求したユーザプロセスに対してシグナル（ユーザプロセスレベルでのソフトウェア割込み）が配送されていないかどうか調べる（ステップ１０３）。この検査はチャネル構造体のプロセスポインタ１４３により４を参照することにより行なわれる。もしシグナルが配送されているときは、このチャネルにおけるデータ転送を中断するが、その際エラーが発生したのと同じ扱いとする。すなわち、このチャネルに対応するチャネル構造体のフラグ１４１にフラグCHERRORをセットし、待機処理部７４中で休眠状態となっているユーザプロセスを走行可能状態とする（ステップ１１２）。もしシグナルが配送されていなければ、次にこのチャネルおいてデータを入力することが可能であるかどうかを、チャネル構造体のデータポインタ１４８、入力データカウンタ１４９、出力データカウンタ１５０、バッファサイズ１５５などをもとに判定する。入力可能かどうかの判定は、バッファにまだ空きがあり、判定の結果、もしそのチャネルに対してデータの入力が可能であれば（ステップ１０５）、入力要求発行処理部７６が呼び出されて、そのチャネルの入力元リソースへのアクセスを行なう入力処理部８に対して入力要求が発行され（ステップ１０６）、変数Ｉの内容に１が加算される（ステップ１０７）。ステップ１０６においてエラーが生じた場合は、既に述べたステップ１１２が実行され、フラグCHERRORがセットされ、そのチャネルを要求しているユーザプロセスを走行可能状態にする。エラーでなければ、次に、このチャネルにおいて出力先リソースにデータを出力することが可能かどうかを判定し（ステップ１０８）、可能であれば出力要求発行処理部７７を呼び出し（ステップ１０９）、さらに変数Ｉの内容に１が加算される（ステップ１１０）。入力の場合と同様、エラーが発生した場合はステップ１１２が実行される（ステップ

１１１）。以上で１つのチャネルに対する処理を終え、チャネル構造体のリストポインタ１４２をみてチャネルリスト上の次のチャネル構造体を取り出す（ステップ１１３）。もし次のチャネル構造体があれば（ステップ１１４）、ステップ１０３に戻る。もしもうチャネル構造体がない場合は、次に変数Ｉが検査される（ステップ１１５）。もし変数Ｉが０でなければ、ステップ１０１に戻ってチャネルリスト６の走査が繰り返される。ここで走査の繰り返しが行なわれるのは、あるチャネルの入出力が、別のチャネルに対して入出力要求が発行されたことによって可能となる可能性があるからである。一方、もし変数Ｉが０、すなわちチャネルリスト６の一連の操作において入力要求あるいは出力要求が発行されたチャネルが１つもなければ、スケジュール処理部７５を終えて、自分を呼び出したルーチンに戻る（ステップ１１６）。

【００６７】（入力要求発行処理部７６）この動作は図１０の通りである。この処理はある１つのチャネルを引数としてスケジュール処理部７５から呼び出される。まず、チャネル構造体の入力記述子１４５とプロセスポインタ１４３から、このチャネルに対応するユーザプロセスの記述子テーブル１８とＯＳ３のカーネルファイルテーブル５が参照され、そのチャネルに対応する入力処理部８が決定される（ステップ１６１）。次に、その入力処理部が同期的であれば（ステップ１６２）、チャネル構造体のフラグ１４１にフラグCHSYNCRDをセットして、待機処理部７４にて休眠中であったユーザプロセスを走行可能状態とする（ステップ１６４）。非同期的であれば、データの入力が終了した時に入力終了処理部７８を呼び出すように設定した後、入力処理部８を呼び出してデータ入力要求を発行する（ステップ１６３）。その後、発行した入力要求に対して既にデータ入力がなされた（例えば入力処理部がファイルシステムで、必要とするデータが既にバッファキャッシュに読み込まれていた場合などが考えられる）かどうかが調べられ（ステップ１６５）、もし読み込まれていれば入力終了処理部７９を呼び出してチャネル構造体の内容を更新する（ステップ１６６）。ここで、入力終了処理部７９を呼び出すときに、入力要求発行処理部７６からの呼び出しであることが入力終了処理部７９において認識できるようにする。入力終了処理部７９は、入力要求発行処理部７６からの呼び出しである場合は、スケジュール処理部７５を呼び出さずに終わる。これにより、スケジュール処理部７５が自分を再帰的に呼び出す結果となってしまうのを防ぐ。その後ステップ１６３または１６４においてエラーが発生したかどうかを調べ（ステップ１６７）、エラーの有無を通知する形で自分を呼び出したスケジュール処理部７５に制御を返す（ステップ１６８、１６９）。

【００６８】（出力要求発行処理部７７）このルーチンの動作は、出力記述子１４６が参照され、この記述子が

指定する出力先リソースに対応する出力処理部９に出力要求が発行されること以外は、既に述べた入力要求発行処理部７６と同じ動作である。

【００６９】（入力終了処理部７８）この動作を図１１に示す。この処理部は、あるチャネルにおいてデータの入力が終了した時に呼び出される。まず、どのチャネルに対する入力が終了したのかが決定され、そのチャネルに対応するチャネル構造体が取り出される（ステップ２０１）。次に、そのチャネルの入力元リソースに対して行なわれたデータ入力がエラーであったかどうかが検査され（ステップ２０２）、もしエラーであればチャネル構造体のフラグ１４１にフラグCHERRORをセットする（ステップ２０３）。その後、実際に読み込まれたデータのバイト数に応じて、チャネル構造体内のバッファ管理構造体１５０を更新する（ステップ２０４）。次に、そのチャネルの入力元リソースから入力すべきデータがこれ以上ないか、あるいはユーザが（トランスファシステムコールの引数として）要求したバイト数だけ入力し終えたかどうかが調べられ（ステップ２０５）、もしなければチャネル構造体のフラグ１４１にフラグCHEOPをセットする（ステップ２０６）。この後、もし入力終了処理部が入力要求発行処理部７６からの呼び出しでなければ（ステップ２０７）、スケジュール処理部７５を呼び出す（ステップ２０８）。それから自分を呼び出したルーチンに戻る（ステップ２０９）。

【００７０】（出力終了処理部７９）この動作を図１２に示す。この処理部は、あるチャネルにおいてデータの出力が終了した時に呼び出される。入力終了処理部７８の場合と同様、まずどのチャネルに対する入力が終了したのかが決定され、そのチャネルに対応するチャネル構造体が取り出される（ステップ２３１）。次に、そのチャネルの出力先リソースに対して行なわれたデータ出力がエラーであったかどうかが検査され（ステップ２３２）、もしエラーであればチャネル構造体のフラグ１４１にフラグCHERRORをセットする（ステップ２３３）。次に、実際に出力先リソースに出力されたデータのバイト数に応じて、チャネル構造体内のバイトカウンタ１４７及びバッファ管理構造体１５０を更新する（ステップ２３４）。ここで、もしチャネル構造体のフラグ１４１にフラグCHEOPがセットされており、かつバッファにデータがない場合（ステップ２３５）あるいはバイトカウンタ１４７が０になってユーザの要求したバイト数だけデータを転送し終えた場合は（ステップ２３６）、チャネル構造体のフラグ１４１にフラグCHDONEをセットする（ステップ２３７）。その後、もし自分が出力要求発行処理部７７から呼び出されたものでなければ（ステップ２３８）、スケジュール処理部７５を呼び出す（ステップ２３９）。その後自分を呼び出したルーチンに戻る（ステップ２４０）。

【００７１】なお、本実施例によれば、データ転送の高

速化に付随して、次のような利点もある。のOSに適合するアプリケーションプログラムを新規に開発する場合には、本実施例で用いた新規なシステムコールを用いる必要がある。しかし、これらのシステムコールは、「データの経路を登録し、そこにデータを流す」という形式の、データの移動経路を明確に記述するので、アプリケーションプログラム内でのデータの移動経路の使用状況が明確になる。このため、そのようなアプリケーションプログラムの開発を容易にする。

【0072】<変形例>実施例1を以下のように変形することも出来る。

【0073】（1）実施例1では、あるチャネルにおけるデータ転送を実行する際に呼び出される入力処理部8や出力処理部9は、ユーザプロセスがそのチャネルを登録する時に何を入出力先リソースとして指定したかによって異なるが、これらのルーチンは従来のものを変更なく使うことができる。しかし、カーネル空間内のバッファ10にデータを入力し、それをまた出力するという処理を、すべてOS3内で行なうという構造になっていることを活用して、既存の入力処理部や出力処理部を、データ転送の際に最適な処理が行なわれるように拡張することも可能で、これによりさらにデータ転送の性能を改善することができる。また、データ転送専用の入力処理部又は出力処理部を本実施例のOSに新規に追加してもよく、そうすればデータ転送をより高機能のものにできる。

【0074】（2）ユーザが独自の入力処理部あるいは出力処理部をプログラムしてOSに登録できるようにすれば、実施例1のOSを適用して処理の性能を高めることのできる範囲が、データ転送だけでなく、ユーザが指定した入力元リソースから、ユーザが指定した出力先リソースへの、ユーザが定義したデータ処理を間に置いたデータ転送にまで、すなわち、ほとんどのアプリケーションプログラムにまで、拡大できる。

【0075】（3）実施例1では単一のプロセッサの下で動作するOSに実装することを想定して説明した。しかし、実施例1は同様の特徴を持つOSに容易に適用可能であり、また疎結合あるいは密結合マルチプロセッサシステムの下で動作するOSに対しても、異なるプロセッサ間の協調のための機能がOSにサポートされていれば、少々の拡張により実施例1を適用することができる。

【0076】<実施例2>実施例1では、そこ用いたチャネルシステムコールの形式から分かるように、記述子を割り当てられるリソース（例えばファイル）のみを、データ転送の入力元リソース又は出力先リソースとして指定できる。本実施例では、記述子を割り当てることができるリソースの種類を増やすことにより、実施例1のデータ転送機能の適用範囲を拡大する。例えば、OS3にユーザプロセスのユーザアドレス空間（ユーザ空間）

のある領域に記述子を割り当てる機能がある場合を考える。この場合、従来のリードシステムコールまたはライトシステムコールによって行なっていた入出力は、チャネルシステムコールとトランスファシステムコールの組み合わせですべて置き換えることができる（すなわちすべての入出力をデータ転送としてとらえることができる）。ユーザ空間の一部に記述子を割り当てることができると、頻繁に入力や出力の行なわれるメモリ空間に記述子をあらかじめ割り当ててチャネルを登録しておくことにより、アドレスに依存しない入出力が可能となる。また、仮想記憶をサポートしているOSでは、ユーザ空間との入出力を行なう場合、一般にユーザアドレス空間内の目的とする領域を物理メモリに固定する処理をそのたびに必要とするが、この処理はオーバヘッドが大きい。そこで、頻繁に入出力を行なうメモリ領域に記述子を割り当てておき、その際にその領域を物理メモリに固定したままにしておくことにより、毎回の入出力（データ転送）においてはこのオーバヘッドを回避することができる。

【0077】<実施例3>実施例1では、あるチャネルが登録された時点で、データ転送すべき入力元リソースと出力先リソースの両方がOSに分かっている。そこで、本実施例では実施例1の転送処理部2（とりわけ入力要求発行処理部76）の拡張により、そのチャネルにおけるデータ転送を最適化する。例えば、あるチャネルにおける入力元リソースと出力先リソースが同じファイルシステムによって管理された2つのファイルである場合、このチャネルのバッファとしてファイルシステムのバッファキャッシュを用いるようにすれば、このチャネルに対してはデータ転送用のバッファ10をカーネル空間内に別に割り当てる手間を省くことができ、かつ、本来必要ないメモリ間コピーの回数を減らして、データ転送をより高速に行なえる。また、ディスプレイのフレームバッファのように、一定のアドレス空間内の領域へのアクセスがハードウェア的に機器へのアクセスへ変換されるような入出力先リソースが、あるチャネルにおける入力あるいは出力先リソースである場合、そのアドレスの領域をそのチャネルのバッファとすることにより、メモリ間コピーの回数だけでなく処理量をも削減することが可能である。さらに、本変形例を実装したOSが動作する計算機にしかるべき手段が備えられている場合は、入力元リソースと出力先リソースの組み合わせによっては、入力元リソースから一度主記憶中にデータを読み込まずに、出力先リソースにハードウェア的に直接出力することができ、それによってより高速なデータ転送を行なえる。

【0078】実施例1の入力要求発行処理部76において、ステップ161ではそのチャネルに対応する入力処理部のアドレスを得る。これを拡張して、このステップにて出力処理部のアドレスも得るようにし、2つのアド

レスの組を見て上記のような転送の最適化ができるかどうか調べるようにする。これは、例えばそうした最適化が可能なアドレスの組（同じファイルシステムの入力と出力のためのアドレスなどの組）をテーブルにして、ＯＳ内の適当な領域に格納しておき、これを参照することによって行うことができる。調べた結果最適化が可能ならば、ステップ１６２以降で呼び出す入力処理部を、別の、最適化された転送のための処理部への呼び出しに変更する。この処理部は前記の最適化が可能なアドレスの組ごとに個別に存在し、それぞれの入力元リソースと出力先リソースの組に応じた最適なデータ転送を実行する。

【００７９】＜実施例４＞実施例１では、チャネルシステムコールがユーザプロセスに返すチャネル識別子を、記述子の１つとして扱うことができるため、チャネルシステムコールの引数として、別のチャネルのチャネル識別子を指定できる。そこで、本実施例では、然るべき処理の実装により、データ転送路の分岐や合流などといった、より複雑な転送経路の設定をユーザプロセスが行えるようにする。例えば、データ転送路の合流を実現する場合、次のように実施例１を拡張すればよい。実施例１の出力要求発行処理部７７では、ある１つのチャネル（ｃとする）のチャネル構造体１３の出力記述子１４３を検査する。このステップを拡張して、検査の結果、出力記述子が別のチャネル（ｄとする）のチャネル識別子であった場合に、そのチャネルｄのチャネル構造体に格納された出力記述子に対応する出力先リソースに出力するようにする。これにより、チャネルｄの出力先リソースには、チャネルｃの入力元リソースからのデータとチャネルｄの入力元リソースからのデータが交互に出力され、結果として２つの入力元リソースからのデータを合流させ、１つの出力先リソースに出力することが可能になる。また、このやり方だと、さらに別のチャネルの出力先リソースとしてチャネルｃ又はチャネルｄを指定することにより、３つ以上のチャネルの合流、すなわち３つ以上の入力元リソースからのデータを合流させることもできる。データの分岐も、実施例１の入力要求発行処理部７６をこれと同様に拡張することにより容易に実現することが出来る。

【００８０】＜実施例５＞実施例１においては、トランスファシステムコールの引数を、チャネルを指定するチャネル識別子と、転送すべきバイト数の２つとしたが、本実施例では、この他にデータ転送の様式を指定するようなオプションを付け加える（あるいは、別のシステムコールでオプションの指定を行なっても良い）。こうしたオプションとしては、例えばデータ転送をできる限り高速に行なわせるのではなく、それよりも遅いある一定の転送速度を指定して、その速度でデータ転送を行なわせる（いわゆるアイソクロナス（isochronous）データ転送）、など様々なものが考えられる。このような時間を意識した機能を実現するためには、ＯＳ３内の各処理部が利用可能な計時機能を計算機システム３００内に設け、実施例１の転送処理部２、とりわけスケジュール処理部７５に、然るべきアルゴリズムを実装すればよい。このような計時機能は、現在ほとんどの計算機システムに備わっている。また、スケジュール処理部７５に実装すべきアルゴリズムとしては、リアルタイムスケジューリングアルゴリズム、またはマルチメディアスケジューリングアルゴリズムとして、従来の技術の項で挙げたGovindan他の文献など、過去に多数発表されている。これらのアルゴリズムは、わずかな変更を加えるだけでスケジュール処理部７５に実装することができる。

【００８１】

【発明の効果】本発明では、アプリケーションプログラムがＯＳにデータ転送を要求するためにＯＳに対して発行されるシステムコールの回数が削減され、それによりこのデータ転送に伴うオーバヘッドが低減する。

【００８２】さらに、こうしたデータ転送が同時に複数実行されているような場合、本発明ではこれらのデータ転送のスケジュールをＯＳにより容易に行い得る。

【図面の簡単な説明】

【図１】本発明によるデータ転送方法を実現するオペレーティングシステムの概略構造を示す図である。

【図２】従来のオペレーティングシステムにおけるデータ転送方法を示す図である。

【図３】本発明の一実施例におけるチャネル構造体のデータ構造を表す図である。

【図４】上記実施例における転送処理部のプログラム構造を示す図である。

【図５】上記実施例におけるチャネル登録処理部の手順を示す流れ図である。

【図６】上記実施例における転送開始／終了処理部の手順を示す流れ図である。

【図７】上記実施例における待機処理部の手順を示す流れ図である。

【図８】上記実施例におけるスケジュール処理部の手順の一部を示す流れ図である。

【図９】上記スケジュール処理部の手順の残部を示す流れ図である。

【図１０】上記実施例における入力要求発行処理部の手順を示す流れ図である。

【図１１】上記実施例における入力終了処理部の手順を示す流れ図である。

【図１２】上記実施例における出力終了処理部の手順を示す流れ図である。

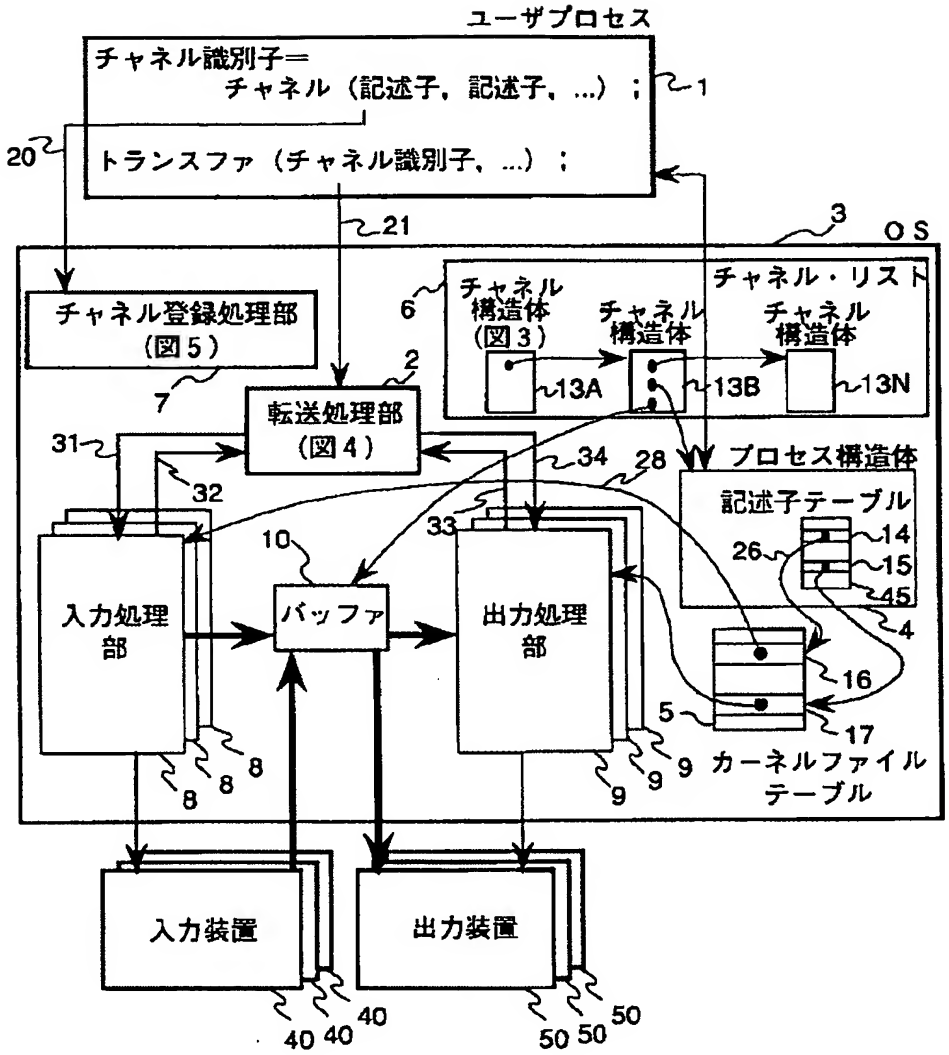
【図１３】従来技術によるデータ転送方法を示す図である。

【符号の説明】

２０：チャネルシステムコール、２１：トランスファシステムコール

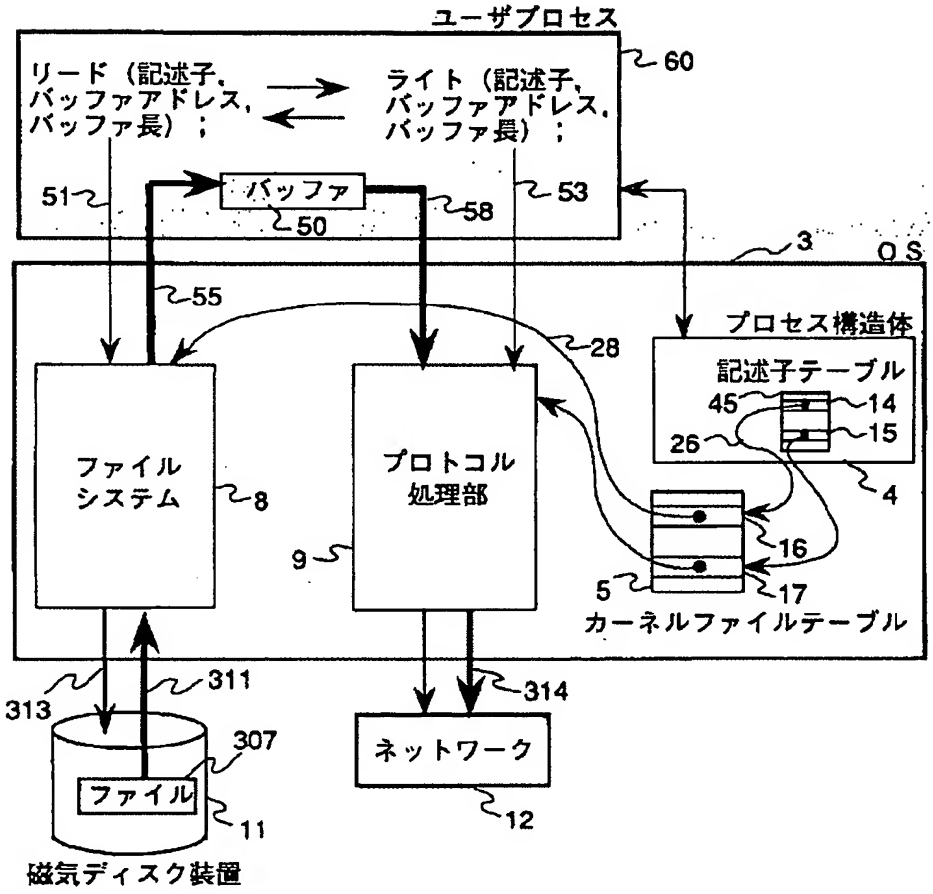
【図１】

図１


ユーザプロセス
チャネル識別子＝
チャネル（記述子，記述子，...）；
トランスファ（チャネル識別子，...）；
1
20
21
3
OS
チャネル登録処理部
（図５）
7
6
チャネル
構造体
（図３）
チャネル
構造体
チャネル
構造体
チャネル・リスト
13A
13B
13N
2
転送処理部
（図４）
31
32
34
28
プロセス構造体
記述子テーブル
14
15
45
26
4
10
33
入力処理
部
バッファ
出力処理
部
16
5
17
8
8
8
9
9
9
カーネルファイル
テーブル
入力装置
出力装置
40
40
40
50
50
50

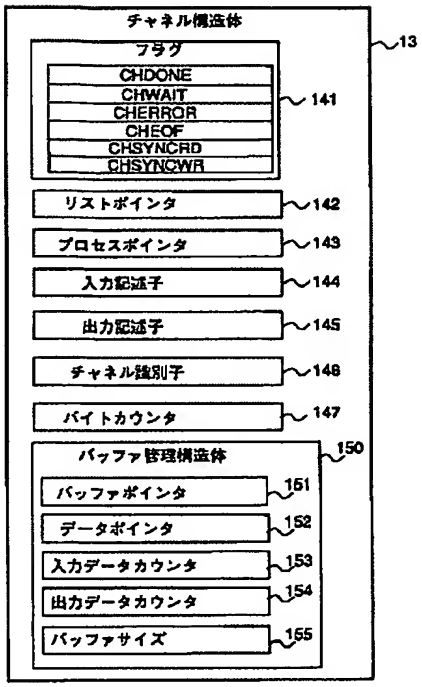
【図2】

図2


ユーザプロセス
60
リード（記述子，
バッファアドレス，
バッファ長）；
ライト（記述子，
バッファアドレス，
バッファ長）；
51
バッファ
50
58
53
3
OS
55
28
プロセス構造体
記述子テーブル
45
14
15
26
4
ファイル
システム
8
プロトコル
処理部
9
16
5
17
カーネルファイルテーブル
313
311
314
307
ファイル
11
ネットワーク
12
磁気ディスク装置

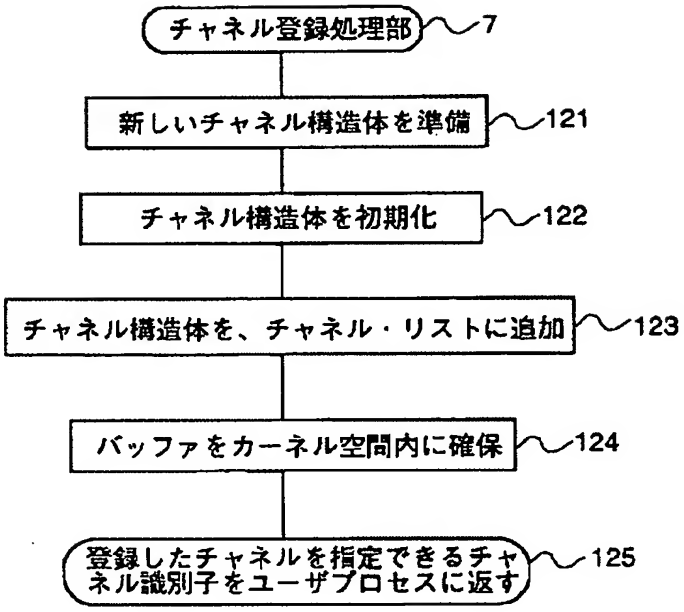
【図３】

図３



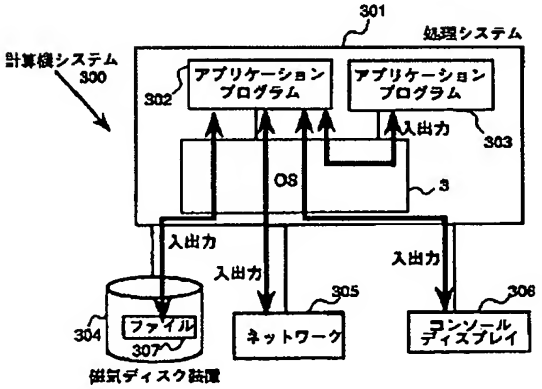
【図５】

図５



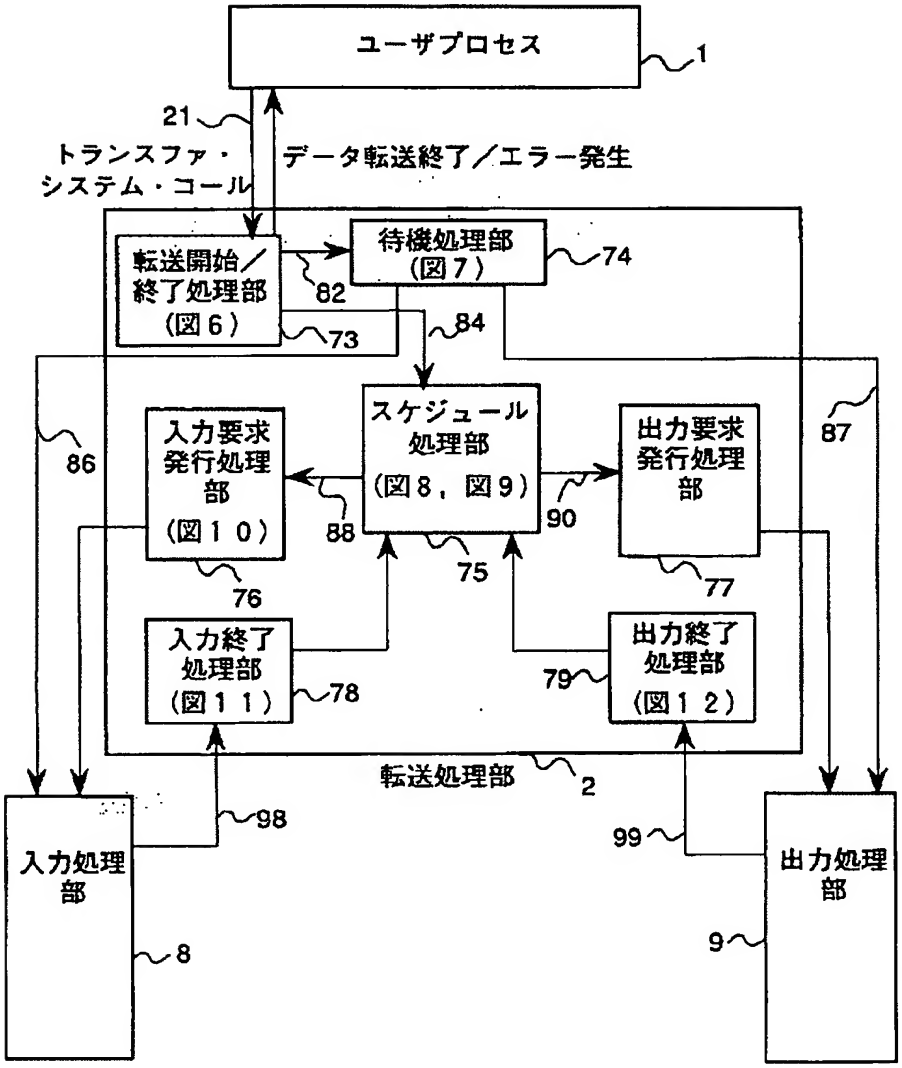
【図１３】

図１３



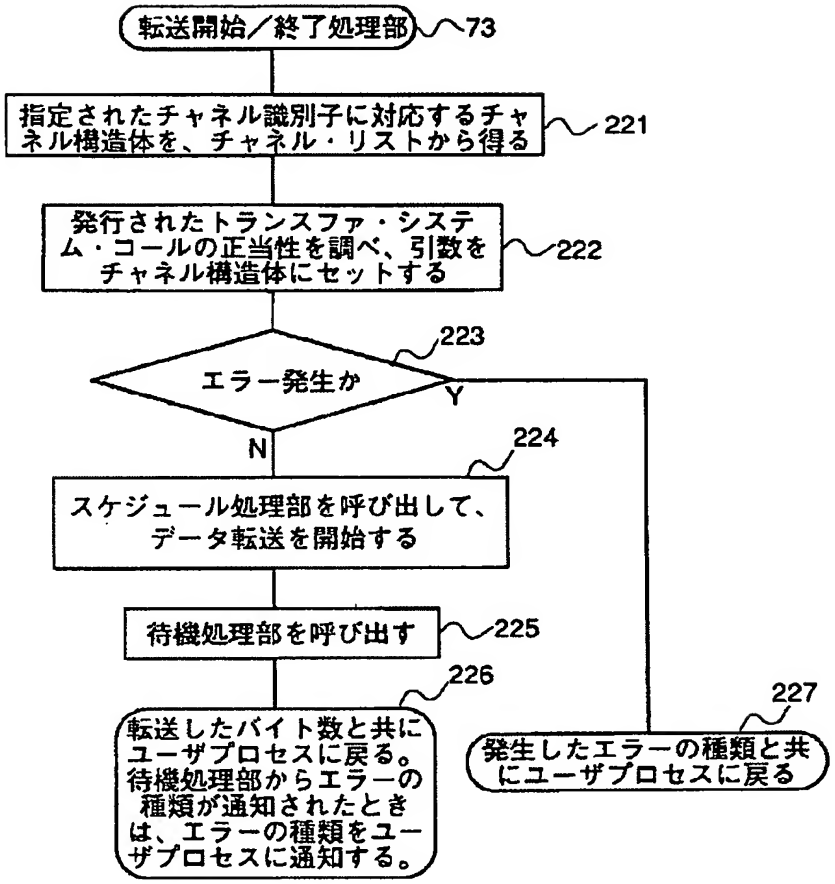
【図４】

図４


ユーザプロセス
1
21
トランスファ・
システム・コール
データ転送終了／エラー発生
転送開始／
終了処理部
（図６）
73
82
待機処理部
（図７）
74
84
スケジュール
処理部
（図８，図９）
入力要求
発行処理
部
（図１０）
76
88
90
出力要求
発行処理
部
77
75
86
87
入力終了
処理部
（図１１）
78
出力終了
処理部
（図１２）
79
転送処理部
2
98
99
入力処理
部
8
出力処理
部
9

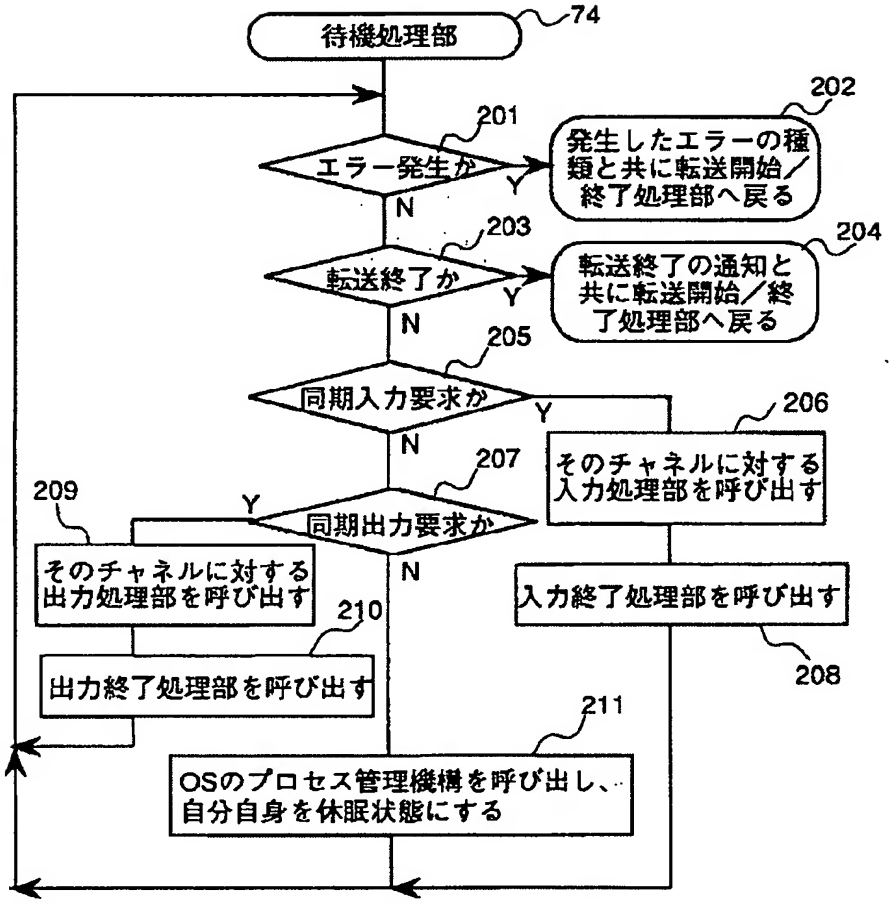
【図６】

図６


転送開始／終了処理部
73
指定されたチャネル識別子に対応するチャネル構造体を、チャネル・リストから得る
221
発行されたトランスファ・システム・コールの正当性を調べ、引数をチャネル構造体にセットする
222
223
エラー発生か
Y
N
224
スケジュール処理部を呼び出して、データ転送を開始する
待機処理部を呼び出す
225
226
転送したバイト数と共にユーザプロセスに戻る。待機処理部からエラーの種類が通知されたときは、エラーの種類をユーザプロセスに通知する。
227
発生したエラーの種類と共にユーザプロセスに戻る

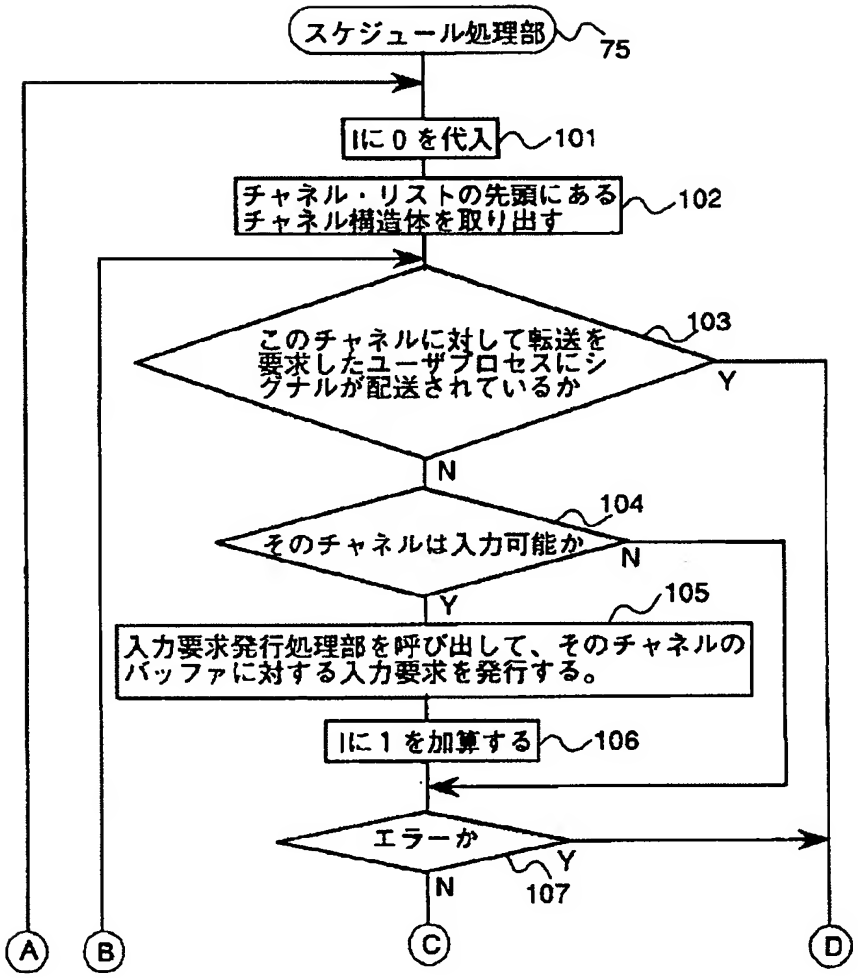
【図７】

図７


待機処理部
74
201
エラー発生か
Y
202
発生したエラーの種類と共に転送開始／終了処理部へ戻る
N
203
転送終了か
Y
204
転送終了の通知と共に転送開始／終了処理部へ戻る
N
205
同期入力要求か
Y
206
そのチャネルに対する入力処理部を呼び出す
N
207
同期出力要求か
Y
209
そのチャネルに対する出力処理部を呼び出す
N
入力終了処理部を呼び出す
208
210
出力終了処理部を呼び出す
211
OSのプロセス管理機構を呼び出し、自分自身を休眠状態にする

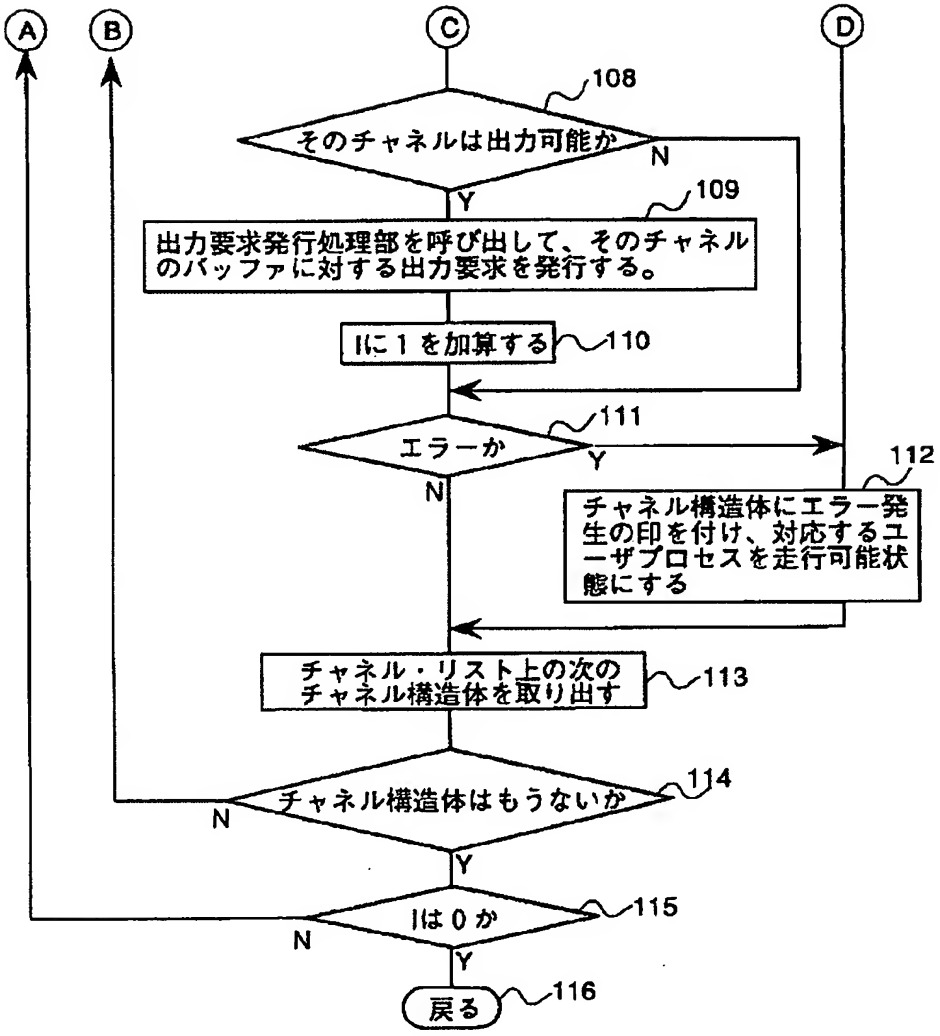
【図8】

図8


スケジュール処理部
75
Iに0を代入
101
チャネル・リストの先頭にある
チャネル構造体を取り出す
102
このチャネルに対して転送を
要求したユーザプロセスにシ
グナルが配送されているか
103
Y
N
そのチャネルは入力可能か
104
N
Y
入力要求発行処理部を呼び出して、そのチャネルの
バッファに対する入力要求を発行する。
105
Iに1を加算する
106
エラーか
Y
107
N
A
B
C
D

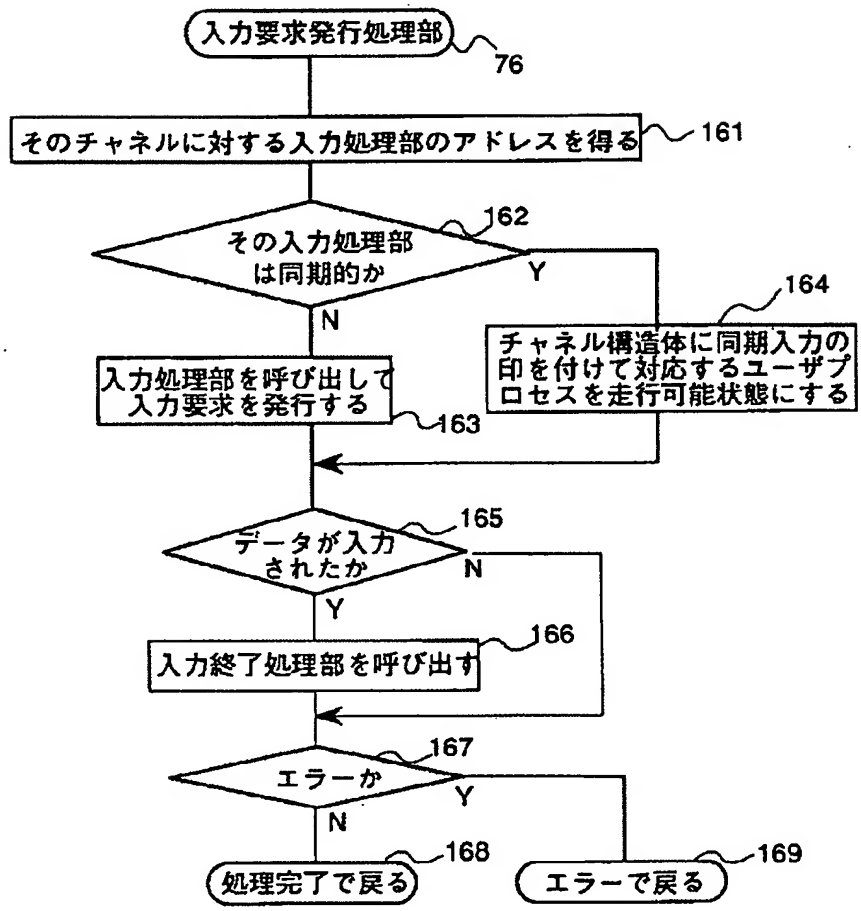
【図９】

図９


A
B
C
D
108
そのチャネルは出力可能か
N
Y
109
出力要求発行処理部を呼び出して、そのチャネルのバッファに対する出力要求を発行する。
Iに１を加算する
110
111
エラーか
Y
N
112
チャネル構造体にエラー発生の印を付け、対応するユーザプロセスを走行可能状態にする
チャネル・リスト上の次のチャネル構造体を取り出す
113
114
チャネル構造体はもうないか
N
Y
115
Iは０か
N
Y
戻る
116

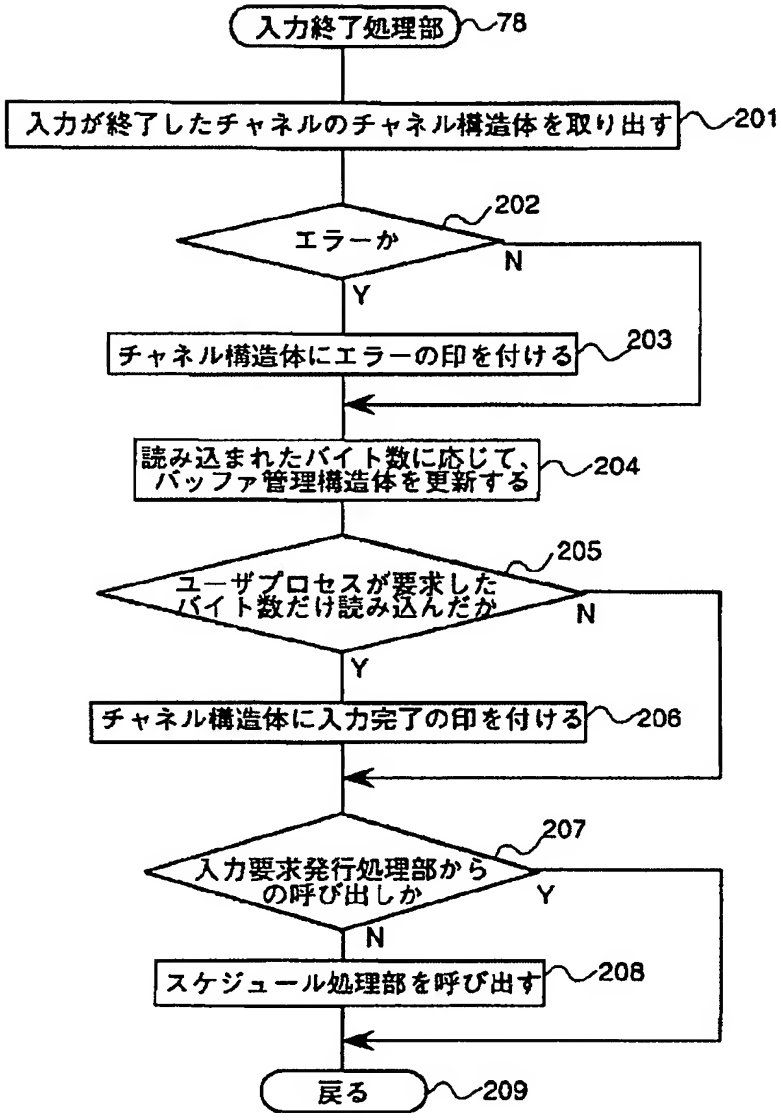
【図10】

図10


入力要求発行処理部
76
そのチャネルに対する入力処理部のアドレスを得る
161
その入力処理部は同期的か
162
Y
N
チャネル構造体に同期入力の印を付けて対応するユーザプロセスを走行可能状態にする
164
入力処理部を呼び出して入力要求を発行する
163
データが入力されたか
165
N
Y
入力終了処理部を呼び出す
166
エラーか
167
Y
N
処理完了で戻る
168
エラーで戻る
169

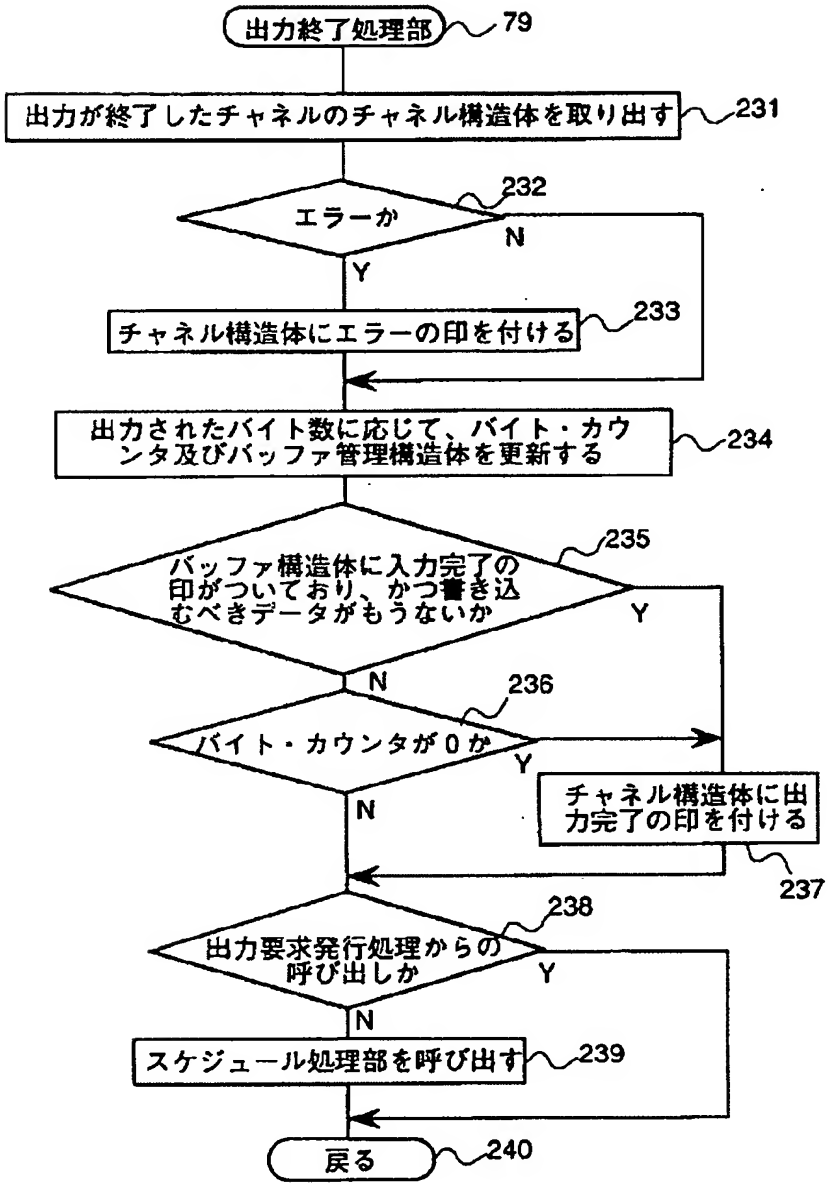
【図１１】

図１１


入力終了処理部
78
入力が終了したチャネルのチャネル構造体を取り出す
201
エラーか
202
N
Y
チャネル構造体にエラーの印を付ける
203
読み込まれたバイト数に応じて、バッファ管理構造体を更新する
204
ユーザプロセスが要求したバイト数だけ読み込んだか
205
N
Y
チャネル構造体に入力完了の印を付ける
206
入力要求発行処理部からの呼び出しか
207
Y
N
スケジュール処理部を呼び出す
208
戻る
209

【図12】

図12


出力終了処理部
79
出力が終了したチャネルのチャネル構造体を取り出す
231
エラーか
232
N
Y
チャネル構造体にエラーの印を付ける
233
出力されたバイト数に応じて、バイト・カウンタ及びバッファ管理構造体を更新する
234
バッファ構造体に入力完了の印がついており、かつ書き込むべきデータがもうないか
235
Y
N
バイト・カウンタが0か
236
Y
N
チャネル構造体に出力完了の印を付ける
237
出力要求発行処理からの呼び出しか
238
Y
N
スケジュール処理部を呼び出す
239
戻る
240